Volume 16 Number 3

ACTA CYBERNETICA

Szeged, 2004

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the University of Szeged, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 2004 Numbers 3–4 of Volume 16 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, University of Szeged, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-546-396, Fax:(36)-(62)-546-397.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/kutatas/actacybernetica/.

L. Budach
University of Postdam
Department of Computer Science
Am Neuen Palais 10
14415 Postdam, Germany

H. Bunke
Universität Bern
Institut für Informatik und
angewandte Mathematik
Längass strasse 51.
CH-3012 Bern, Switzerland

B. Courcelle
Universitè Bordeaux-1
LaBRI, 351 Cours de la Libèration
33405 TALENCE Cedex
France

J. Demetrovics
MTA SZTAKI
Budapest, Lágymányosi u. 11.
H-1111 Hungary

B. Dömölki
IQSOFT
Budapest, Teleki Blanka u. 15-17.
H-1142 Hungary

J. Engelfriet
Leiden University
LIACS
P.O. Box 9512, 2300 RA Leiden
The Netherlands

Z. Ésik
University of Szeged
Department of Foundations of
Computer Science
Szeged, Aradi vértanúk tere 1.
H-6720 Hungary

L. Lovász
Eötvös Loránd University
Department of Computer Science
Budapest, Kecskeméti u. 10-12.
H-1053 Hungary

G. Păun
Institute of Mathematics
Romanian Academy
P.O.Box 1-764, RO-70700
Bucuresti, Romania

A. Prékopa
Eötvös Loránd University
Department of Operations Research
Budapest, Kecskeméti u. 10-12.
H-1053 Hungary

A. Salomaa
University of Turku
Department of Mathematics
SF-20500 Turku 50, Finland

L. Varga
Eötvös Loránd University
Department of General Computer Science
Budapest, Pázmány Péter sétány 1/c.
H-1117 Hungary

H. Vogler
Dresden University of Technology
Department of Computer Science
Foundations of Programming
D-01062 Dresden, Germany

G. Wöginger
Department of Matematics
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands

Acta Cybernetica 16 (2004) 367–384.

A Uniform Approach to Test Computational Complementarity

Elena Calude, Bruce Mills, and Lan Mills*

Abstract

Studies of computational complementarity properties in finite state interactive automata may shed light on the nature of both quantum and classical computation. But, complementarity is *difficult* to test even for small-size automata. This paper introduces the concept of an observation graph of an automaton which is used as the main tool for the design of an algorithm which tests, in a uniform manner, two types of complementarity properties. Implementations have been run on a standard desktop computer examining all 5-state binary automata.

1 Two Computational Complementarity Principles

Building on Moore's "Gedanken" experiments, in [15, 14] complementarity was modeled by means of finite automata. Two new computational complementarity principles have been introduced and studied in [3, 6, 5, 4, 2] using Moore's automata.

To understand Moore's approach it is enough, at this stage, to say that the machines we are going to consider are *finite* in the sense that they have a finite number of states, a finite number of input symbols, and a finite number of output symbols. Such a machine has a strictly deterministic behaviour: the current state of the machine depends only on its previous state and previous input; the current output depends only on the present state. A (simple) Moore experiment can be described as follows: a copy of the machine will be experimentally observed, i.e. the experimenter will input a finite sequence of input symbols to the machine and will observe the sequence of output symbols. The correspondence between input and output symbols depends on the particular chosen machine and on its initial state. The experimenter will study the sequences of input and output symbols and will try to conclude that "the machine being experimented on was in state q at the beginning of the experiment".[1] Moore's experiments have been studied

*Institute for Information and Mathematical Sciences, Massey University at Albany, Private Bag 102904, NSMC, Auckland, New Zealand.
Email: {E.Calude,B.I.Mills,L.Mills}@massey.ac.nz

[1]This is often referred to as a *state identification experiment.*

from a mathematical point of view by various researchers, notably by Ginsburg [9], Chaitin [7], Conway [8], and Brauer [1]. A comprehensive survey on testing finite state machines is presented in [11].

In what follows we are going to use two non-equivalent concepts of computational complementarity based upon modeling finite automata (see [3]). Informally, they can be expressed as follows. Consider the class of all elements of reality[2] and consider the following properties.

A Any two distinct elements of reality can be mutually distinguished by a suitably chosen measurement procedure.

B For any element of reality, there exists a measurement which distinguishes between this element and all the others. That is, a distinction between any one of them and all the others is operational.

C There exists a measurement which distinguishes between any two elements of reality. That is, a single pre-defined experiment exists to distinguish between an arbitrary pair of elements of reality. (Classical case.)

Complementarity corresponds to the following cases:

CI Property **A** *but not property* **B** *(and therefore not* **C***):* The elements of reality can be mutually distinguished by experiments, but one of these elements cannot be distinguished from all the other ones by any single experiment.

CII Property **B** *but not property* **C***:* Any element of reality can be distinguished from all the other ones by a single experiment, but there does not exist a single experiment which distinguishes between any pair of distinct elements.

2 Moore Automata

A finite deterministic automaton consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from some fixed alphabet. For each symbol there is exactly one transition out of each state, possible back to the state itself. So, formally, a *finite automaton* consists of a finite set Q of states, an input alphabet Σ, and a transition function $\delta : Q \times \Sigma \to Q$. Sometimes a fixed state, say 1, is considered to be the *initial state*, and a subset F of Q denotes the *final states.* A *Moore automaton* is a finite deterministic automaton having an *output function* $f : Q \to O$, where O is a finite set of output symbols. At each time the automaton is in a given state q and is continuously emitting the output $f(q)$. The automaton remains in state q until it receives an input signal σ, when it assumes the state $\delta(q, \sigma)$ and starts emitting $f(\delta(q, \sigma))$. In this paper we are going to concentrate on the case of automata on a binary alphabet $\Sigma = \{0, 1\}$ having $O = \Sigma$. So, from now on, a Moore automaton will be just a triple $M = (Q, \delta, f)$. Let Σ^* be the set of all finite sequences (words) over the alphabet Σ, including the empty word

[2]The terms "elements of reality", "properties", and "observables" will be used as synonyms.

ϵ (the neutral element in the semigroup of string concatenation); by Σ^+ we denote $\Sigma^* \setminus \{\epsilon\}$. The transition function δ can be extended to a function $\overline{\delta} : Q \times \Sigma^* \to Q$, as follows: $\overline{\delta}(q, \epsilon) = q, \overline{\delta}(q, \sigma w) = \overline{\delta}(\delta(q, \sigma), w), \forall q \in Q, \sigma \in \Sigma, w \in \Sigma^*$. The output produced by an experiment started in state q with input sequence $w \in \Sigma^*$ is described by $E(q, w)$, where E is the function $E : Q \times \Sigma^* \longrightarrow \Sigma^*$ defined as follows: $E(q, \varepsilon) = f(q)$, $E(q, \sigma w) = f(q)E(\delta(q, \sigma), w), q \in Q, \sigma \in \Sigma, w \in \Sigma^*$, and $f : Q \longrightarrow O(= \Sigma)$ is the output function. Consider, for example, Moore's automaton, in which $Q = \{1, 2, 3, 4\}$, $\Sigma = \{0, 1\}$. The transition is given by the following tables

q	σ	$\delta(q, \sigma)$
1	0	4
1	1	3
2	0	1
2	1	3

q	σ	$\delta(q, \sigma)$
3	0	4
3	1	4
4	0	2
4	1	2

Table 1.

and the output function is defined by $f(1) = f(2) = f(3) = 0$, $f(4) = 1$. The following graphical representation will be consistently used in what follows:

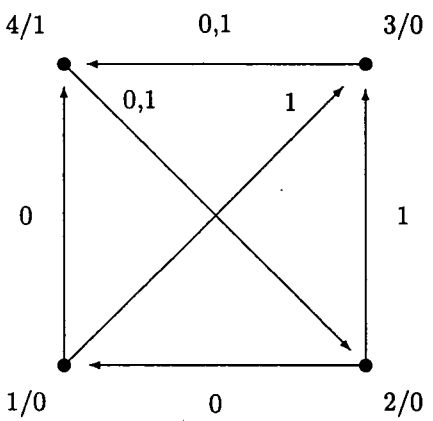

Figure 1.

The experiment starting in state 1 with input sequence 000100010 leads to the output 0100010001. Indeed, $E(1, 000100010) = f(1)f(4)f(2)f(1)f(3)f(4)f(2)f(1)f(3)f(4) = 0100010001$.

From a mathematical point of view properties **A**, **B**, **C** can be expressed as follows. Let $M = (Q, \delta, f)$ be a Moore automaton. Following Moore [13] we shall say that a state q is "indistinguishable" from a state q' (with respect to M) if every experiment performed on M starting in state q produces the same outcome as it would starting in state q'. Formally, $E(q, x) = E(q', x)$, for all words $x \in \Sigma^+$. A pair of states will be said to be "distinguishable" if they are not "indistinguishable".

- The automaton M has property **A** if every pair of different states of M are distinguishable, i.e. for every distinct states q, q' there exists a word $w \in \Sigma^+$ (depending upon q, q') such that $E(q, w) \neq E(q', w)$. This is simply the assertion that the automaton is minimal.

- The automaton M has property **B** if every state of M is distinguishable from any other distinct state, i.e. for every state q there exists a word $w \in \Sigma^+$ (depending upon q) such that $E(q, w) \neq E(q', w)$, for every state q' distinct from q.

- The automaton M has property **C** if there exists an experiment distinguishing between each different states of M, i.e. there exists a word $w \in \Sigma^+$ such that $E(q, w) \neq E(q', w)$, for every distinct states q, q'.

Of course, **C** implies **B**, which, in turn, implies **A**; none of the converse implications is true, hence we get *CI*, *CII*.

We continue with some examples of Moore automata having **C**, *CI*, and *CII*.

First, the automaton in Figure 2 has **C** as experiment 10 distinguishes between any pair of distinct states.

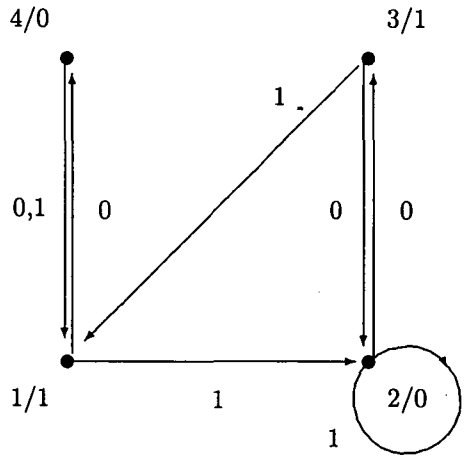

Figure 2.

Moore's automaton in Figure 1 has **A** but non-**B**, hence *CI* (cf. [13]). Every pair of distinct states can be distinguished by an experiment: states $1,2$ by $x = 0$, states $1,3$ by $x = 1$, states $1,4$ by $x = 0$, states $2,3$ by $x = 0$, states $2,4$ by $x = 0$, and states $3,4$ by $x = 0$. However, there is no (unique) experiment capable to distinguish between every pair of arbitrary distinct states. If the experiment starts with 1, i.e. $x = 1u$, then $E(1,x) = E(2,x)$, that is x cannot distinguish between the states $1,2$ as $E(1,x) = E(1,1u) = f(1)f(\delta(1,1))E(\delta(1,1),u) = f(1)f(3)E(3,u) = 00E(3,u)$ and $E(2,x) = E(2,1u) = f(2)f(\delta(2,1))E(\delta(2,1),u) = f(2)f(3)E(3,u) = 00E(3,u)$. If the experiment starts with 0, i.e. $x = 0v$, $v \in \Sigma^*$, then x cannot distinguish between the states $1,3$ as $E(1,x) = E(1,0v) = f(1)f(\delta(1,0))E(\delta(1,0),v) = f(1)f(4)E(4,v) = 01E(4,v)$ and $E(3,x) = E(3,0v) = f(3)f(\delta(3,0))E(\delta(3,0),v) = f(3)f(4)E(4,v) = 01E(4,v)$.

The automaton in Figure 3 has **B** but not **C**, hence *CII*. Indeed, the following pairs of states are distinguishable by every experiment: $(1,2)$, $(1,4)$, $(2,3)$, $(3,4)$. Accordingly, 1 is distinguishable from the other states by $w = 0$, 2 is distinguishable by $w = 1$, 3 is distinguishable by $w = 0$ and 4 is distinguishable by $w = 1$, so the automaton has property **B**. It does not have property **C** because any experiment w which starts with 1, i.e. $w = 1x$, $x \in \Sigma^*$, does not distinguish between 1 and 3, and any experiment w which starts with 0, i.e. $w = 0y$, $y \in \Sigma^*$, does not distinguish between 2 and 4.

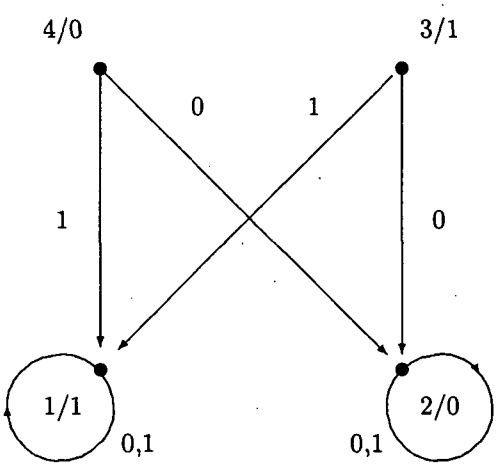

Figure 3.

3 An Algorithm for Testing Simultaneously *CI* and *CII*

In this section we briefly review a few facts on partitions and present the algorithm that is used to test properties **A**, **B** and **C**, which uses partitions defined on sets of states of an automaton. An elegant algebraic theory for machine decomposition based on the closed partition lattice of a machine is presented in [10] and efficient algorithms for constructing the lattice are presented in [12]; here we construct a different partition lattice testing the properties *CI* and *CII*, a different problem.

A partition P of a set Q is a set of non-empty disjoint sets whose union is Q. Partitions are in an one-to-one onto correspondence with equivalence relations. In particular, we will use the partition induced by the level sets of a map $f : Q \to Q$, that is, the sets $[q]_f = \{x : f(x) = f(q)\}, q \in Q$.

Given two partitions P_1 and P_2 of Q, we say that P_1 is no coarser than (or at least as coarse as) P_2, written $P_1 \leq P_2$ if for every $p_1 \in P_1$, there exists $p_2 \in P_2$ such that $p_1 \subseteq p_2$. We say that P_2 is coarser than P_1 if $P_1 \leq P_2$ and $P_1 \neq P_2$, in symbols $P_1 < P_2$. The term *finer* means the inverse relation of coarser. When $P_1 \leq P_2$ we say that P_1 is a refinement of P_2, or that P_2 is a coarsening of P_1.

Treating the above refinement relation as a partial order $\leq$, we see that the greatest lower bound $P_1 \wedge P_2$ is the coarsest partition of Q that is a refinement of both P_1 and P_2. This operation, which we will call *CCR* (the coarsest common refinment), can be conducted in principle by taking the intersection of all classes in P_1 with all classes in P_2, and then throwing out the empty sets.

Let P_1 and P_2 be two partitions of Q, and $\equiv_1$ and $\equiv_2$ be the corresponding equivalences. Then the equivalence relation $p \equiv q$ defined by $p \equiv_1 q$ and $p \equiv_2 q$, corresponds to $P_1 \wedge P_2$.

The level sets of the composition $f \circ g$ are coarser than those of g; if g is invertible then the level sets are the same. Let $f \times g : a \to (f(a), g(a))$. Then, the level set partition of $f \times g$ is the coarsest common refinement of f and g.

For an automaton M, we construct a graph, called an observation graph, which describes how information about the state of machine changes with observations. Each vertex R is a record $\begin{bmatrix} t \\ P \end{bmatrix}$, where the two fields t and P are the configuration of states under the transition function and the partition induced by the output function respectively. The partition induced by t, $\Pi(t)$, is given by the following equivalence relation: i is equivalent to j modulo $\Pi(t)$ if $f(t[i]) = f(t[j])$.

An edge $\left(\begin{bmatrix} t_1 \\ P_1 \end{bmatrix}, \begin{bmatrix} t_2 \\ P_2 \end{bmatrix} \right)$ belongs to the graph exactly when there exists $s \in \{0, 1\}$ such that $t_2 = \delta(t_1, s)$ and $P_2 = \Pi(f \circ t_2) \wedge P_1$. Since the CCR of two partitions is no coarser than either it is apparent that along any path through the graph the partitions may become finer. The function $\left(\begin{bmatrix} t \\ P \end{bmatrix}, s \right) \to P$, mapping vertices into lattice of partitions, is monotonic.

If at the start of a path the condition $P_1 \leq t_1$ occurs, then, $P_2 = \Pi(\delta(t_1, s)) \wedge$

$P_1 \leq \Pi(\delta(t_1, s)) = \Pi(t_2)$.

For any path $\left(\begin{bmatrix} t_1 \\ P_1 \end{bmatrix} \ldots \begin{bmatrix} t_n \\ P_n \end{bmatrix}\right)$ in the observation graph, $P_n \leq P_1$. For any path $\left(\begin{bmatrix} t_1 \\ P_1 \end{bmatrix} \ldots \begin{bmatrix} t_n \\ P_n \end{bmatrix}\right)$ in the observation graph if $P_1 \leq \Pi(t_1)$, then $P_n = P_1$. Finally, suppose that $\left(\begin{bmatrix} t \\ P \end{bmatrix}, B\right)$ is a rooted sub-tree of the observation graph, and $P \leq \Pi(t)$. The partition P is constant throughout the entire tree since each node on the rooted sub-tree is the end point of a path starting at the root $\begin{bmatrix} t \\ P \end{bmatrix}$. Consequently, the node $\begin{bmatrix} t \\ P \end{bmatrix}$ will be pruned (ignored).

The algorithm for testing properties **A**, **B**, and **C** for an automaton M generates records that are nodes of an observation graph and checks whether the partitions components of the nodes verify the conditions associated with the properties **A**, **B**, and **C**. The algorithm has the following steps.

Step 1. Initialization of

- a vector *Counter* recording all non-repeating nodes generated so far
- a trimmed binary tree *OG* recording the nodes in the observational graph
- a vector *TB* recording those states for which the condition in property **B** is currently verified, and a Boolean variable *hasB* that is true if M has property **B** and false otherwise
- a table *TA* recording those states for which the condition in property **A** is currently verified, and a Boolean variable *hasA* that is true if M has property **A** and false otherwise.

Step 2. Generate and test the first record

- *Step 2.1.* Generate the first record. The first record, say R, will be the root of *OG*. Its two components are given by
  - the vector of states $(1, 2, \ldots, n)$ and
  - the partition P_R, generated by the output function f,

  so $R = \begin{bmatrix} (1, 2, \ldots, n) \\ P_R \end{bmatrix}$
- *Step 2.2.* If M has **C** stop. Else:
- *Step 2.3.* Add the record to *Counter*
- *Step 2.4.* Update *TA* and *hasA*

- *Step 2.5.* Update TB and $hasB$
- *Step 2.6.* Add the record to OG

[Comment: We generate (from left to right) all children of non-pruned nodes. If no child can be generated, we check the values of $hasB$ and $hasA$ to determine whether the automaton has **B** or **A**, then stop.]

Step 3. While there are children to be generated do

- *Step 3.1.* Generate the next record obtained from the left/right child, say $N = \begin{bmatrix} t_N \\ P_N \end{bmatrix}$. Its two components are given by
  - t_N the vector of states $(i_1, i_2, \ldots, i_n)$, obtained by applying the transition function on each element of sequence of states in the parent node with input letter 0 for the left child and 1 for the right child and
  - the partition P_N obtained by taking the CCR of the parent's partition component and the partion of states induced by the output function on t_N
- *Step 3.2.* If N is in *Counter*, then go to *Step 3.1.* Otherwise, add the current record to *Counter*
- *Step 3.3.* If M has **C**, stop. Else:
- *Step 3.4.* Update TA and $hasA$
- *Step 3.5.* Update TB and $hasB$
- *Step 3.6.* If the record can be pruned, then go to *Step 3.1*
- *Step 3.6.* Add the node N to OG and go to *Step 3.1*

End of while loop.

If TA is false, then return non-**A**, stop; else, if TB is false, then return *CI*, stop; else, retun *CII*, stop.

End of algorithm.

4 The Algorithm in Action

In this section we present some examples illustrating the algorithm presented in the previous section.

Example 1. Let us run the algorithm on the automaton in Figure 1.

Step 1. Initialize OG to $\emptyset$, *Counter* to the $\emptyset$ vector, TA and TB, respectively, the $n \times n$ matrix with all elements 0 and the n-element all 0 vector, and the Boolean variables *hasA* and *hasB* to false

Step 2 Generate the first record

Step 2.1 Generate the record $R = \begin{bmatrix} 1, 2, 3, 4 \\ \{1,2,3\},\{4\} \end{bmatrix}$

Step 2.2 The record R does not satisfy $\mathbf{C}$ as $P_R \neq \{1\},\{2\},\{3\},\{4\}$

Step 2.3 Update *Counter* to $[R]$

Step 2.4 Update TA to $\begin{bmatrix} 0, 0, 0, 1 \\ 0, 0, 0, 1 \\ 0, 0, 0, 1 \\ 1, 1, 1, 0 \end{bmatrix}$ and *hasA* to false

Step 2.5 Update TB to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ and *hasB* to false

Step 2.6 Update OG to $(R, \emptyset)$

Step 3 Generate the next child

First iteration:

Step 3.1 The next record, $N0$, is

$$\begin{bmatrix} 4, 1, 4, 2 \\ \{1,3\},\{2\},\{4\} \end{bmatrix},$$

where $t_{N0} = (4, 1, 4, 2)$ is obtained by applying the transition function on $1, 2, 3, 4$ to the input letter 0. The partition induced by the output function on $4, 1, 4, 2$ is $\{1,3\},\{2,4\}$. Taking the CCR between this partition and the partition component of its parent, i.e. $P_R = \{1,2,3\},\{4\}$, we obtain the partition component of $N0$. Therefore

$$P_{N0} = \{1,3\},\{2,4\} \wedge \{1,2,3\},\{4\} = \{1,3\},\{2\},\{4\}$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0]$$

Step 3.3 The current automaton has not **C** as

$$P_{N0} \neq \{1\}, \{2\}, \{3\}, \{4\}$$

Step 3.4 Update TA to $\begin{bmatrix} 0,1,0,1 \\ 1,0,1,1 \\ 0,1,0,1 \\ 1,1,1,0 \end{bmatrix}$ and $hasA$ to false

Step 3.5 Update TB to $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$ and $hasB$ to false

Step 3.6 As

$$\{1,3\}, \{2\}, \{4\} \wedge \{1,3\}, \{2\}, \{4\} = \{1,3\}, \{2\}, \{4\}$$

it follows that

$$\{1,3\}, \{2,4\} \wedge P_{N0} = P_{N0}$$

and therefore this record has to be pruned (as none of its children can bring any new information)

Step 3 Generate next child

Second iteration:

Step 3.1 The next record, $N1$ is

$$\begin{bmatrix} 3,3,4,2 \\ \{1,2\}, \{3\}, \{4\} \end{bmatrix},$$

where $t_{N1} = (3,3,4,2)$ is obtained by applying the transition function on $1,2,3,4$ and the input letter 1. The partition induced by the output function on $3,3,4,2$ is $\{1,2,4\}, \{3\}$. Taking the CCR between this partition and the partition component of the node's parent, i.e. $P_R = \{1,2,3\}, \{4\}$, we obtain the partition component of $N1$. Therefore

$$P_{N1} = \{1,2,4\}, \{3\} \wedge \{1,2,3\}, \{4\} = \{1,2\}, \{3\}, \{4\}$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0, N1]$$

Step 3.3 The current automaton has not **C** as

$$P_{N1} \neq \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update TA to $\begin{bmatrix} 0,1,1,1 \\ 1,0,1,1 \\ 1,1,0,1 \\ 1,1,1,0 \end{bmatrix}$ and $hasA$ to true

Step 3.5 Update TB to $\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ and $hasB$ to false

Step 3.6 As $\{1,2\},\{3\},\{4\} \wedge P_N = \{1,2\},\{3\},\{4\} \wedge \{1,2\},\{3\},\{4\} = \{1,2\},\{3\},\{4\}$ this record has to be pruned

Step 3 Generate the next child

Third iteration:

As $OG = (R, \emptyset)$ and $Counter = (R, N0, N1)$, there is no child to generate, stop. As $hasA$ is true and $hasB$ is false, the output is "the automaton has CI".

Example 2. Let us run the algorithm on the automaton in Figure 2.

Step 1. Initialize OG to $\emptyset$, $Counter$ to the $\emptyset$ vector, TA and TB, respectively, the $n \times n$ matrix with all elements 0 and the n-element all 0 vector, and the Boolean variables $hasA$ and $hasB$ to false

Step 2 Generate the first record

Step 2.1 Generate the record $R = \begin{bmatrix} 1,2,3,4 \\ \{1,3\},\{2,4\} \end{bmatrix}$

Step 2.2 As $P_R \neq \{1\},\{2\},\{3\},\{4\}$ the automaton has not **C**

Step 2.3 Update $Counter = [R]$

Step 2.4 Update TA to $\begin{bmatrix} 0,1,0,1 \\ 1,0,1,0 \\ 0,1,0,1 \\ 1,0,1,0 \end{bmatrix}$ and $hasA$ to false

Step 2.5 Update TB to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and $hasB$ to false

Step 2.6 Update OG to $(R, \emptyset)$

Step 3 Generate the next child

First iteration:

Step 3.1 The next record, $N0$, is

$$\begin{bmatrix} 4,3,2,1 \\ \{1,3\},\{2,4\} \end{bmatrix}$$

where $t_{N0} = (4,3,2,1)$ is obtained by applying the transition function on $1,2,3,4$ to the input letter 0. The partition induced by the output function on $4,3,2,1$ is $\{1,3\},\{2,4\}$. Taking the CCR between this partition and the partition component of its parent $PR = \{1,3\},\{2,4\}$, we obtain the partition component of $N0$:

$$P_{N0} = \{1,3\},\{2,4\} \wedge P_R = \{1,3\},\{2,4\}$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0]$$

Step 3.3 The current automaton has not **C** as

$$P_{N0} = \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update TA to $\begin{bmatrix} 0,1,0,1 \\ 1,0,1,0 \\ 0,1,0,1 \\ 1,0,1,0 \end{bmatrix}$ and $hasA$ to false

Step 3.5 Update TB to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and $hasB$ to false

Step 3.6 As

$$\{1\},\{2\},\{3\},\{4\} \wedge P_{N0} \neq P_{N0}$$

this record should not be pruned. Update $OG = (R, N0, [R, N0])$

Step 3 Generate the next child

Second iteration:

Step 3.1 The next record, $N1$, is

$$\begin{bmatrix} 2,2,1,1 \\ \{1\},\{2\},\{3\},\{4\} \end{bmatrix}$$

where $t_{N1} = (2,2,1,1)$ is obtained by applying the transition function on states $1,2,3,4$ and the input letter 1. The partition induced by the output function on $2,2,1,1$ is $\{1,2\},\{3,4\}$. Taking the CCR between this partition and the partition component of the node's parent, i.e. $P_R = \{1,3\},\{2,4\}$, we obtain the partition component of $N1$:

$$P_{N1} = \{1,3\},\{2,4\} \wedge \{1,2\},\{3,4\} = \{1\},\{2\},\{3\},\{4\}$$

Step 3.2 The current node is not in *Counter*, so we add it:

$$Counter = [R, N0, N1]$$

Step 3.3 The current automaton has **C** as

$$P_{N1} = \{1\},\{2\},\{3\},\{4\}$$

so the output is "the automaton has **C**".

Example 3. Let us run the algorithm on the automaton in Figure 3.

Step 1. Initialize OG to $\emptyset$, *Counter* to the $\emptyset$ vector, TA and TB, respectively, the $n \times n$ matrix with all elements 0 and the n-element all 0 vector, and the Boolean variables *hasA* and *hasB* to false

Step 2 Generate the first record

Step 2.1 Generate the record $R = \begin{bmatrix} 1,2,3,4 \\ \{1,3\},\{2,4\} \end{bmatrix}$

Step 2.2 The record R has not **C** as $P_R \neq \{1\},\{2\},\{3\},\{4\}$

Step 2.3 Update $Counter = [R]$

Step 2.4 Update TA to $\begin{bmatrix} 0,1,0,1 \\ 1,0,1,0 \\ 0,1,0,1 \\ 1,0,1,0 \end{bmatrix}$ and *hasA* to false

Step 2.5 Update TB to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and $hasB$ to false

Step 2.6 Update OG to $(R, \emptyset)$

Step 3 Generate the next child

First iteration:

Step 3.1 The next record, $N0$, is

$$\begin{bmatrix} 1,2,2,2 \\ \{1\},\{2,4\},\{3\} \end{bmatrix},$$

where $t_{N0} = (1,2,2,2)$ is obtained by applying the transition function on states $1,2,3,4$ to the input letter 0. The partition induced by the output function on $1,2,2,2$ is $\{1\},\{2,3,4\}$. Taking the CCR between this partition and the partition component of its parent $P_R = \{1,3\},\{2,4\}$ we obtain the partition component of $N0$:

$$P_{N0} = \{1\},\{2,3,4\} \wedge P_R = \{1\},\{2,4\},\{3\}$$

Step 3.2 The current node is not in *Counter*, so we add it:

$$Counter = [R, N0]$$

Step 3.3 The current automaton has not **C** as

$$P_{N0} \neq \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update TA to $\begin{bmatrix} 0,1,1,1 \\ 1,0,1,0 \\ 1,1,0,1 \\ 1,0,1,0 \end{bmatrix}$ and hasA to false

Step 3.5 Update TB to $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ and $hasB$ to false

Step 3.6 As

$$\{1\},\{2,3,4\} \wedge P_{N0} = P_{N0}$$

this record has to be pruned

Step 3 Generate the next child

Second iteration:

Step 3.1 The next record, $N1$, is

$$\begin{bmatrix} 1, 2, 1, 1 \\ \{1,3\},\{2\},\{4\} \end{bmatrix},$$

where $t_{N1} = (1, 2, 1, 1)$ is obtained from applying the transition function on states $1, 2, 3, 4$ and the input letter 1. The partition induced by the output function on $2, 1, 1, 1$ is $\{1,3,4\},\{2\}$. Taking the CCR between this partition and the partition component of the node's parent $P_R = \{1,3\},\{2,4\}$ we obtain the partition component of $N1$:

$$P_{N1} = \{1,3,4\},\{2\} \wedge \{1,3\},\{2,4\} = \{1,3\},\{2\},\{4\}.$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0, N1]$$

Step 3.3 The current automaton has not **C** as

$$P_{N1} \neq \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update TA to $\begin{bmatrix} 0, 1, 1, 1 \\ 1, 0, 1, 1 \\ 1, 1, 0, 1 \\ 1, 1, 1, 0 \end{bmatrix}$ and $hasA$ to true

Step 3.5 Update TB to $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ and $hasB$ to true

Step 3.6 As $\{1,3,4\},\{2\} \wedge P_N = P_N$ this record has to be pruned

Step 3 Generate the next child

Third iteration:

As $OG = (R, \emptyset)$ and $Counter = [R, N0, N1]$, there is no child to generate, stop. As $hasA$ is true and $hasB$ is true, the automaton has property **B**, but not **C**, so the output is "the automaton has *CII*".

5 Experimental Results

The proposed algorithm was implemented in C and the program[3] was run on a Pentium III, i686 processor using Redhat 8.0 Linux, 250 Mb of RAM. The aim was to study the distributions of *CI* and *CII* over the set of all possible automata with a given number of states and input/output symbols. Table 2 presents the results of the main tests that have been done so far.

$n \times s$	# automata	time (sec)	*CI*	*CII*	*CII/CI* %	*CI* %	*CII* %
2×2	32	< 1	0	0	0	0	0
3×2	2916	< 2	0	0	0	0	0
4×2	524288	< 11	73728	30720	41.67	14.06	5.86
5×2	156250000	< 8435	46862400	19436160	41.47	29.99	12.44
4×3	452984832	< 14018	54577152	46227456	84.70	12.05	10.12

Table 2.

In the first column $n \times s$ stands for the class of automata with n states and s input letters. Because of symmetries (the automaton (Q, δ, f) "is equivalent" to the automaton $(Q, \delta, 1 - f)$), the program actually tests only half of automata of type $n \times s$; the numbers of tested automata are shown in the second column. The third column contains the time for processing all automata mentioned in the second column. The numbers of automata verifying *CI* and *CII* are given in the next two columns. The last three columns present the percentage of *CII* over *CI* and the percentage of *CI*, respectively, *CII*, over the total number of automata processed.

We also tested automata with more than five states. For example, the 10-state automaton in Table 3

q	$\delta(q,0)$	$\delta(q,1)$	$f(q)$
1	2	1	0
2	3	2	0
3	4	3	0
4	5	4	0
5	6	5	0
6	7	6	0
7	8	7	0
8	9	8	0
9	9	10	0
10	10	10	1

Table 3.

[3] See `http://www.massey.ac.nz/~bimills/obgraph.c` for the program.

has **A** (00000001 distinguishes the pairs (i,j) for $i = 1,2,3, j = 1,2,\ldots,10$ and $i \neq j$, 00001 distinguishes the pairs $(4,5),(4,6),(4,7),(4,8),(4,9)$, 0001 distinguishes the pairs $(5,6),(5,7),(5,8),(5,9)$, 001 distinguishes the pairs $(6,7),(6,8),(6,9)$, 01 distinguishes the pairs $(7,8),(7,9)$, 1 distinguishes the pair $(8,9)$ and ε distinguishes the pairs $(i,10)$ for $i = 1,2,\ldots,9$), has **B** (as state 1 is distinguished from all other states by the word 00000001, state 2 by 000000101, state 3 by 00000101, state 4 by 0000101, state 5 by 000101, state 6 by 00101, state 7 by 0101, state 8 by 101, state 9 by 1, and state 10 by ε) and has **C** (the word 101010101010101 distinguishes every pair of distinct states). The algorithm has scanned 766 nodes in less than a second.

6 Final Remarks

Based on the concept of observation graph of an automaton, new equivalent definitions have been given for two types of computational complementarity studied in [3]. As a result, we proposed an algorithm for simultaneously determining these properties. The algorithm has been shown in practice to be fast enough (on a standard desktop machine) for testing all binary Moore machines up to five states. Some other experiments reported in the paper illustrate the power of the algorithm.

Many problems remain open; for example, what is the complexity of the decision problems *CI, CII.*

Acknowledgement

We are grateful to the anonymous referee for suggestions which improved the paper.

References

[1] Brauer, W. *Automaten theorie.* Teubner, Stuttgart, 1984.

[2] Calude, C. S., Calude, E. and Ştefănescu, C. Computational complementarity for Mealy automata, *EATCS Bull.* 66 (1998), 139–149.

[3] Calude, C., Calude, E., Svozil, K. and Yu, S. Physical versus computational complementarity I. *International Journal of Theoretical Physics 36* (1997), 1495–1523.

[4] Calude, C. S. and Lipponen, M. Computational complementarity and sofic shifts, in X. Lin (ed.). *Theory of Computing 98, Proceedings of the 4th Australasian Theory Symposium, CATS'98,* Springer-Verlag, Singapore, 1998, 277–290.

[5] Calude, E. and Lipponen, M. Minimal deterministic incomplete automata. *Journal of Universal Computer Science 11* (1997), 1180–1193.

[6] Calude, E. and Lipponen, M. Deterministic incomplete automata: Simulation, universality and complementarity, in C. S. Calude, J. Casti, and M. J. Dinneen (eds.). *Unconventional Models of Computation*, Springer-Verlag, Singapore, 1998, 131–149.

[7] Chaitin, G. J. An improvement on a theorem by E. F. Moore. *IEEE Transactions on Electronic Computers EC-14* (1965), 466–467.

[8] Conway, J. H. *Regular Algebra and Finite Machines.* Chapman and Hall Ltd., London, 1971.

[9] Ginsburg, S. On the length of the smallest uniform experiment which distinguishes the terminal states of the machine. *Journal of the Association for Computing Machinery 5* (1958), 266–280.

[10] Hartmanis J. and Stearns R. E. *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[11] Lee, D. and Yannakakis, M. Principles and methods of testing finite state machines—A survey, *Proc. IEEE* 84, 8, (1996),1089–1123.

[12] Lee, D. and Yannakakis, M. Closed partition lattice and machine decomposition, *IEEE Transactions on Computers*, 51, 2 (2002), 216–228.

[13] Moore, E. F. Gedanken-experiments on sequential machines, in C. E. Shannon and J. McCarthy (eds.). *Automata Studies*, Princeton University Press, Princeton, 1956.

[14] Schaller, M. and Svozil, K. Automaton partition logic versus quantum logic. *International Journal of Theoretical Physics 34*, 8 (1995), 1741–1750.

[15] Svozil, K. *Randomness & Undecidability in Physics.* World Scientific, Singapore, 1993.

Received September, 2003

Acta Cybernetica **16** (2004) 385–397.

Two-Way Metalinear PC Grammar Systems and Their Descriptional Complexity

Alexander Meduna*

Abstract

Besides a derivation step and a communication step, a two-way PC grammar system can make a reduction step during which it reduces the right-hand side of a context-free production to its left hand-side. This paper proves that every non-unary recursively enumerable language is defined by a centralized two-way grammar system, Γ, with two metalinear components in a very economical way. Indeed, Γ's master has only three nonterminals and one communication production; furthermore, it produces all sentential forms with no more than two occurrences of nonterminals. In addition, during every computation, Γ makes a single communication step. Some variants of two-way PC grammar systems are discussed in the conclusion of this paper.

1 Introduction

Over the past few years, the formal language theory has intensively investigated many variants of PC grammar systems (see [12]), which consist of several simultaneously working and communicating components, represented by grammars. This paper introduces another variant of this kind, called *two-way PC grammar systems*, which make three kinds of computational steps—derivation, reduction, and communication. More precisely, a two-way PC grammar system, Γ, makes a derivation step as usual; that is, it rewrites the left-hand side of a production with its right-hand side. During a reduction step, however, Γ rewrites the right-hand side with the left hand-side. Finally, Γ makes a communication step in a usual PC-grammar-system way; in addition, however, after making this step, it changes the computational way from derivations to reductions or vice versa.

As reduction steps represent a mathematically natural modification of derivation steps, a discussion of two-way PC grammar systems surely deserves our attention from a theoretical viewpoint. From a practical viewpoint, this discussion is important as well. Indeed, two-way PC grammar systems actually formalize computational units combining both reduction and derivation steps, which frequently occur in applied computer science. To give some specific examples, consider, for

*Department of Information Systems, Faculty of Information Technology, Brno University of Technology, Božetěchova 2, Brno 61266, Czech Republic

instance, compilers. A parser is often written so it actually represents a combination of a bottom-up parser for expressions and a top-down parser for general program flow. While the former makes reductions, the latter makes derivations; as a whole, the parser thus makes both. To give another example in this area, the three-address code generation often consist of top-down syntax-directed generation of abstract syntax tree followed by a bottom-up translation of this tree to the desired three-address code. Again, both reductions and derivations take part in this translation process as a whole. As a result, there surely exist both theoretically and pragmatically sound reasons for investigating two-way PC grammar systems.

This paper narrows its attention to the centralized two-way metalinear PC grammar systems working in a non-returning mode. That is, since they are centralized, only their first components, called the masters, can cause these systems to make a communication step. Since they are metalinear, all their components are represented by metaliner grammars. Finally, as they work in a non-returning mode, after communicating, their components continue to process the current string rather than return to their axioms. Regarding these systems, the present paper concentrates its discussion on their descriptional complexity because this complexity represents an intensively studied area of today's formal language theory.

As its main result, this paper proves that the centralized two-way metalinear PC grammar systems characterize the family of non-unary recursively enumerable languages in a very economical way. Indeed, every non-unary recursively enumerable language is defined by a centralized two-way grammar system with two metalinear components so that during every computation Γ makes a single communication step. In addition, Γ's three-nonterminal master has only one production with a communication symbol and each of its sentential forms contains no more than two occurrences of nonterminals. In the conclusion of this paper, some terminating and parallel variants of these two-way systems are introduced and analogical results to the above characterization are achieved.

2 Preliminaries

This paper assumes that the reader is familiar with the formal language theory (see [9], [14]). For a set, Q, $card(Q)$ denotes the cardinality of Q. For an alphabet, V, V^* represents the free monoid generated by V under the operation of concatenation. The unit of V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, V^+ is thus the free semigroup generated by V under the operation of concatenation. For every $w \in V^*$, $|w|$ denotes the length of w. Furthermore, for every $0 \leq i \leq |w|$ and $L \in V^*$, we introduce the following denotation:

- $\mathbf{length}(L) = \{|w| : w \in L\}$
- $\mathbf{reversal}(w)$ denotes the reversal of w
- $\mathbf{reversal}(L) = \{\mathbf{reversal}(w) : w \in L\}$
- $\mathbf{alph}(w)$ denotes the set of letters occurring in w
- $\mathbf{alph}(L) = \{a : a \in \mathbf{alph}(w) \text{ with } w \in L\}$
- $\mathbf{sym}(w, i)$ denotes the ith symbol in w

- $\mathbf{prefix}(w,i)$ denotes the set of w's prefixes of length i or less
- $\mathbf{prefix}(w) = \mathbf{prefix}(w,|w|)$
- $\mathbf{suffix}(w,i)$ denotes the set of w's suffixes of length i or less
- $\mathbf{suffix}(w) = \mathbf{suffix}(w,|w|)$
- $\mathbf{prefix}(L) = \{x : x \in \mathbf{prefix}(w) \text{ for some } w \in L\}$
- $\mathbf{suffix}(L) = \{x : x \in \mathbf{suffix}(w) \text{ for some } w \in L\}$

For every $W \subseteq V$, $\mathbf{del}(w,W)$ denotes the word resulting from w by the deletion of all symbols from W in w; more formally, $\mathbf{del}(w,W) = \rho(w)$, where ρ is the weak identity over V^* defined as $\rho(b) = \varepsilon$ for every $b \in W$ and $\rho(a) = a$ for every $a \in V - W$. Let $\mathbf{keep}(w,W)$ denote the word resulting from w by the deletion of all symbols from $V - W$ in w; more formally, $\mathbf{keep}(w,W) = \theta(w)$, where θ is the weak identity over V^* defined as $\theta(b) = \varepsilon$ for every $b \in V - W$ and $\theta(a) = a$ for every $a \in W$. For instance, for $w = abac$, $\mathbf{alph}(w) = \{a,b,c\}, \mathbf{prefix}(w,2) = \{\varepsilon, a, ab\}, \mathbf{sym}(w,3) = a, \mathbf{del}(w,\{a\}) = bc, \mathbf{keep}(w,\{a,b\}) = aba$.

A *queue grammar* (see [7]) is a sixtuple, $Q = (V,T,W,F,s,P)$, where V and W are alphabets satisfying $V \cap W = \emptyset, T \subseteq V, F \subseteq W, s \in (V-T)(W-F)$, and $P \subseteq (V \times (W-F)) \times (V^* \times W)$ is a finite relation such that for every $a \in V$, there exists an element $(a,b,x,c) \in P$ for some $b \in W - F, x \in V^*$, and $c \in W$. If $u,v \in V^*W$ such that $u = arb, v = rzc, a \in V, r,z \in V^*, b,c \in W$, and $(a,b,x,c) \in P$, then $u \Rightarrow v\ [(a,b,z,c)]$ in G or, simply, $u \Rightarrow v$. The language of Q, $L(Q)$, is defined as $L(Q) = \{w \in T^* : s \Rightarrow^* wf \text{ where } f \in F\}$.

Now, we slightly modify the notion of a queue grammar. A left-extended queue grammar is a sixtuple, $Q = (V,T,W,F,s,P)$, where V,T,W,F, and s have the same meaning as in a queue grammar. $P \subseteq (V \times (W-F)) \times (V^* \times W)$ is a finite relation (as opposed to an ordinary queue grammar, this definition does not require that for every $a \in V$, there exists an element $(a,b,x,c) \in P$). Furthermore, assume that $\# \notin V \cup W$. If $u,v \in V^*\{\#\}V^*W$ so that $u = w\#arb, v = wa\#rzc, a \in V, r,z,w \in V^*, b,c \in W$, and $(a,b,x,c) \in P$, then $u \Rightarrow v[(a,b,z,c)]$ in G or, simply, $u \Rightarrow v$. In the standard manner, extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, define $\Rightarrow^+$ and $\Rightarrow^*$. The language of Q, $L(Q)$, is defined as $L(Q) = \{v \in T^* : \#s \Rightarrow^* w\#vf$ for some $w \in V^*$ and $f \in F\}$. Less formally, during every step of a derivation, a left-extended queue grammar shifts the rewritten symbol over #; in this way, it records the derivation history, which represents a property fulfilling a crucial role in the proof of Lemma 4 in the next section.

3 Definitions

As already sketched in Section 1, this paper discusses grammar systems (see [1, 2, 3, 4, 5, 7]), concentrating its attention on PC grammar systems (see [6, 11, 12, 13, 15, 16]). The present section introduces a new version of these systems. First, based on two-way k-linear PC components, it defines two-way k-linear n-PC grammar systems. Then, it introduces several notions concerning them. Finally, two examples are given.

Let k and n be two positive integers. A two-way k-linear PC component is a quadruple, $G = (N, T, P, S)$, where N and T are two disjoint alphabets. Symbols in N and T are referred to as nonterminal and terminals, respectively, and $S \in N$ is the start symbol of G. Set $M = N - \{S\}$. P is a finite set of productions such that each $r \in P$ has one of these forms

- $S \to x$, where $x \in (T \cup M)^*$ and x contains no more than k occurrences of symbols from M,
- $A \to x$, where $A \in M$ and $x \in T^*MT^* \cup T^*$.

Let $u, v \in (N \cup T)^*$. For every $A \to x \in P$, write $uAv \ {}_d\Rightarrow \ uxv$ and $uxv \ {}_r\Rightarrow \ uAv$; d and r stand for a direct *d*erivation and a direct *r*eduction, respectively. To express that G makes $uAv \ {}_d\Rightarrow \ uxv$ according to $A \to x$, write $uAv \ {}_d\Rightarrow \ uxv \ [A \to x]$; $uxv \ {}_r\Rightarrow \ uAv \ [A \to x]$ have an analogical meaning in terms of ${}_r\Rightarrow$. A *two-way k-linear n-PC grammar system* is an $n + 1$-tuple

$$\Gamma = (Q, G_1, \ldots, G_n),$$

where $Q = \{q_i : i = 1, \ldots, n\}$, whose members are called query symbols, and for all $i = 1, \ldots, n, G_i = (Q \cup N_i, T, P_i, S_i)$ is a two-way k-linear PC component such that $Q \cap (N_i \cup T) = \emptyset$ (notice that each G_i has the same terminal alphabet, T); let $q\text{-}P_i \subseteq P_i$ denote the set of all productions in P_i containing a query symbol. A configuration is an n-tuple of the form $(x_1, \ldots, x_n)$, where $x_i \in (Q \cup N_i \cup T)^*, 1 \leq i \leq n$. The start configuration, s, is defined as $s = (S_1, \ldots, S_n)$. Let Θ denote the set of all configurations of Γ. For every $x \in \Theta$ and $i = 1, \ldots, n, i\text{-}x$ denotes its ith component—that is, if $x = (x_1, \ldots, x_i, \ldots, x_n)$, then $i\text{-}x = x_i$. For every $x \in \Theta$, define the mapping ${}_x\theta$ over $\{i\text{-}x : 1 \leq i \leq n\}$ as ${}_x\theta(i\text{-}x) = z_1 z_2 \ldots z_{|i\text{-}x|}$ where for all $1 \leq h \leq |i\text{-}x|$,

if for some $q_j \in Q, i = 1, \ldots, n, \mathbf{sym}(i\text{-}x, h) = q_j$ and $\mathbf{alph}(j\text{-}x) \cap Q = \emptyset$, then $z_h = j\text{-}x$; otherwise (that is, $\mathbf{sym}(i\text{-}x, h) \notin Q$ or $\mathbf{alph}(j\text{-}x) \cap Q \neq \emptyset$), $z_h = \mathbf{sym}(i\text{-}x, h)$.

Let $y, x \in \Theta$. Write

- $y \ {}_d\Rightarrow \ x$ in Γ if $i\text{-}y \ {}_d\Rightarrow \ i\text{-}x$ in G_i or $i\text{-}y = i\text{-}x$ with $i\text{-}y, i\text{-}x \in T^*$, for all $i = 1, \ldots, n$;
- $y \ {}_r\Rightarrow \ x$ in Γ if $i\text{-}y \ {}_r\Rightarrow \ i\text{-}x$ in G_i or $i\text{-}y = i\text{-}x$ with $i\text{-}y, i\text{-}x \in \{S_i\} \cup T^*$, for all $i = 1, \ldots, n$;
- $y \ {}_q\Rightarrow \ x$ in Γ if $i\text{-}x = {}_y\theta(i\text{-}y)$ in G_i for all $i = 1, \ldots, n$.

Informally, Γ works in three computational modes—${}_d\Rightarrow, {}_r\Rightarrow, {}_q\Rightarrow$, which symbolically represent a direct *derivation*, *reduction*, and *communication*, respectively. Let $l \geq 1, \alpha_j \in \Theta, 1 \leq i \leq l$, and $\alpha_0 \ {}_{l_1}\Rightarrow \ \alpha_1 \ {}_{l_2}\Rightarrow \ \alpha_2 \ldots \alpha_{l-1} \ {}_{l_l}\Rightarrow \ \alpha_l$ where $l_m \in \{d, r, q\}, 1 \leq m \leq l$; write $\alpha_0 \Rightarrow^* \alpha_l$ if $l_1 = d$ and each $l_p \in \{d, r, q\}, 2 \leq p \leq l - 1$, satisfies:

- if $l_p = q$ then $l_{p+1}, l_{p-1} \in \{d, r\}$ and $l_{p+1} \neq l_{p-1}$,
- if $l_p \in \{d, r\}$ then $l_{p+1} \in \{q, l_p\}$.

Informally, after making a communication step, Γ changes the computational mode from d to r and vice versa; after making a derivation or reduction step, it does not. Consider $\alpha_0 \Rightarrow^* \alpha_l$ that consists of l direct computational steps, $\alpha_0 \ {}_{l_1}\!\Rightarrow \alpha_1 \ {}_{l_2}\!\Rightarrow \alpha_2 \ldots \alpha_{l-1} \ {}_{l_l}\!\Rightarrow \alpha_l$, satisfying the above properties. Set $\kappa(\alpha_0 \Rightarrow^* \alpha_l) = \{\alpha_0, \alpha_1, \ldots, \alpha_l\}$; that is, $\kappa(\alpha_0 \Rightarrow^* \alpha_l)$ denote the set of all configurations occurring in $\alpha_0 \Rightarrow^* \alpha_l$. Furthermore, for each $l = 1, \ldots, n$, set $\kappa(i\text{-}\alpha_0 \Rightarrow^* i\text{-}\alpha_l) = \{i\text{-}\beta : \beta \in \kappa(\alpha_0 \Rightarrow^* \alpha_l)\}$. Finally, for each $h = 1, \ldots, n$, *h-computation*$(i\text{-}\alpha_0 \Rightarrow^* i\text{-}\alpha_l)$ denote $h\text{-}\alpha_0 \ {}_{l_1}\!\Rightarrow h\text{-}\alpha_1 \ {}_{l_2}\!\Rightarrow h\text{-}\alpha_2 \ldots h\text{-}\alpha_{l-1} \ {}_{l_l}\!\Rightarrow h\text{-}\alpha_l$ The *language* of Γ, $L(\Gamma)$, is defined as

$$L(\Gamma) = \{z \in T^* : \sigma \Rightarrow^* \alpha \text{ in } \Gamma \text{ with } z = \mathbf{del}(1\text{-}a, S_1), \text{ for some } \alpha \in \Theta\}.$$

Informally, $L(\Gamma)$ contains $z \in T^*$ if and only if there exists $\alpha \in \Theta$ such that $\sigma \Rightarrow^* a$ in Γ and the deletion of each S_1 in $1\text{-}a$ results in z. A computation $\sigma \Rightarrow^* a$ in Γ with $\mathbf{del}(1\text{-}a, S_1) \in L(\Gamma)$ is said to be successful. By a *two-way metalinear n-PC grammar system*, we refer to any two-way k-linear n-PC grammar system, where $k \geq 1$.

Notice that after communicating, the components of the above systems continue to process the current string rather than return to their axioms. In other words, they work in the non-returning mode (see [7]). The returning mode is not discussed in this paper.

For a two-way k-linear PC grammar system, $\Gamma = (Q, G_1, \ldots, G_n)$, we next introduce some special notions.

Finite index. Let $\sigma \Rightarrow^* x$ be any successful computation in Γ, where $x \in \Theta$, and let $i \in \{1, \ldots, n\}$. By $i\text{-}index(\sigma \Rightarrow^* x)$, we denote the maximum number in $\mathbf{length}(\mathbf{keep}(\kappa(i\text{-}\sigma \Rightarrow^* i\text{-}x), N_i))$. If for every successful computation $\sigma \Rightarrow^* \xi$ in Γ, where $\xi \in \Theta$, there exists $k \geq 1$ such that $i\text{-}index(\sigma \Rightarrow^* \xi) \leq k$, G_i is of a *finite index*. If G_i is of a finite index, $index(G_i)$ denotes the minimum number h satisfying $i\text{-}index(\sigma \Rightarrow^* \xi) \leq h$, for every successful computation $\sigma \Rightarrow^* \varpi$ in Γ, where $\varpi \in \Theta$. By $index(G_i) = \infty$, we express that G_i is not of a finite index. If G_j is of a finite index for all $j = 1, \ldots, n$, Γ is of a *finite index* and $index(\Gamma)$ denotes the minimum number g satisfying $index(G_l) \leq g$, for all $l = 1, \ldots, n$. By $index(\Gamma) = \infty$, we express that Γ is not of a finite index.

q-Degree. For $\sigma \Rightarrow^* x$ in Γ, where $x \in \Theta$, $q\text{-}degree(\sigma \Rightarrow^* x)$ denotes the number of communication steps $({}_q\!\Rightarrow)$ in $\sigma \Rightarrow^* x$. If for every computation $\sigma \Rightarrow^* \xi$ in Γ, where $\xi \in \Theta$, there exists $k \geq 1$ such that $q\text{-}degree(\sigma \Rightarrow^* \xi) \leq k$, Γ is of a *finite q-degree*. If Γ is of a finite q-degree, $q\text{-}degree(\Gamma)$ denotes the minimum number h satisfying $q\text{-}degree(\sigma \Rightarrow^* \xi) \leq h$, for every computation $\sigma \Rightarrow^* \xi$ in Γ; by $q\text{-}degree(\Gamma) = \infty$, we express that Γ is not of a finite q-degree.

Centralized Version. Γ is *centralized* if no query symbol occurs in any production of P_i in $G_i = (N_i, T_i, P_i, S_i)$, for all $i = 2, \ldots, n$. In other words, only P_1 can contain some query symbols, so G_1, called the *master* of Γ, is the only component that can cause Γ to perform a communication step.

This paper concentrates its attention on the centralized version of two-way k-linear 2-PC grammar systems. Therefore, we conclude this section by two examples illustrating these systems.

Example 1. Consider the centralized two-way two-linear 2-PC grammar system, $G = (\{q_1, q_2\}, G_1, G_2)$, where $G_1 = (\{S_1, A, B\}, T, P_1, S_1), G_2 = (\{S_2, B, Y\}, T, P_2, S_2), T = \{a, b, c\}$, $P_1 = \{S_1 \to A, A \to cA, A \to cq_2, Q_2 \to B, B \to q_2, B \to \varepsilon, S_1 \to B\}$, and $P_2 = \{S_2 \to YB, B \to B, Y \to aYb, Y \to ab\}$.

For instance, Γ generates $c^3a^3b^3a^3b^3a^3b^3$ as (S_1, S_2) $_d\Rightarrow$ (A, YB) $_d\Rightarrow$ $(cA, aYbB)$ $_d\Rightarrow$ $(ccA, aaYbbB)$ $_d\Rightarrow$ $(cccq_2, a^3b^3B)$ $_q\Rightarrow$ $(c^3a^3b^3B, a^3b^3B)$ $_r\Rightarrow$ $(c^3a^3b^3q_2, a^3b^3B)$ $_q\Rightarrow$ $(c^3a^3b^3a^3b^3B, a^3b^3B)$ $_d\Rightarrow$ $(c^3a^3b^3a^3b^3q_2, a^3b^3B)$ $_q\Rightarrow$ $(c^3a^3b^3a^3b^3a^3b^3B, a^3b^3B)$ $_r\Rightarrow$ $(c^3a^3b^3a^3b^3a^3b^3S_1, a^3b^3B)$ with $\mathbf{del}(c^3a^3b^3a^3b^3a^3b^3S_1, S_1) = c^3a^3b^3a^3b^3a^3b^3$.

Observe that $L(\Gamma) = \{c^jx^i : x \in H, j, i \geq 1, |x| = 2j\}$, where $H = \{a^nb^n : n \geq 1\}$. Furthermore, notice that $index(G_1) = 1$ and $index(G_2) = 2$, so Γ is of a finite index. On the other hand, $q\text{-}degree(\Gamma) = \infty$.

Example 2. Consider the centralized two-way one-linear 2-PC grammar system $G = (\{q_1, q_2\}, G_1, G_2)$ where $G_1 = (\{S_1, A, B\}, T, P_1, S_1), G_2 = (\{S_2, B\}, T, P_2, S_2), T = \{a, b, c\}$, $P_1 = \{S_1 \to A, A \to aAa, A \to aq_2a, B \to Bc, S_1 \to B\}$, and $P_2 = \{S_2 \to B, B \to bBc\}$.

For instance, Γ makes (S_1, S_2) $_d\Rightarrow$ (A, B) $_d\Rightarrow$ (aAa, bBc) $_d\Rightarrow$ $(aaq_2aa, bbBcc)$ $_q\Rightarrow$ $(aabbBccaa, bbBcc)$ $_r\Rightarrow$ $(aabbBcaa, bBc)$ $_r\Rightarrow$ $(aabbS_1caa, B)$.

Notice that $L(\Gamma) = \{a^nb^nc^ma^n : n \geq m \geq 0\}, index(G_1) = 1, index(G_2) = 1$, and $q\text{-}degree(\Gamma) = 1$.

4 Main Result

This section proves that every non-unary recursively enumerable language is defined by a centralized two-way three-linear 2-PC grammar system, $\Gamma = (\{Q_2\}, G_1, G_2)$, such that $index(G_1) = 2, index(G_2) = 3$, and $q\text{-}degree(\Gamma) = 1$. As a result, $index(\Gamma) = 3$. In addition, its three-nonterminal master, G_1, has only one production containing a query symbol.

Lemma 1. *For every recursively enumerable language, L, there exists a left-extended queue grammar, Q, satisfying $L(Q) = L$.*

Proof. Recall that every recursively enumerable language is generated by a queue grammar (see [8]). Clearly, for every queue grammar, there exists an equivalent left-extended queue grammar. Thus, this lemma holds. ■

Lemma 2. *Let Q' be a left-extended queue grammar. Then, there exists a left-extended queue grammar, $Q = (V, T, W, F, s, R)$, such that $L(Q') = L(Q)$, $W = X \cup Y \cup \{1\}$, where $X, Y, \{1\}$ are pairwise disjoint, and every $(a, b, x, c) \in R$ satisfies either $a \in V - T, b \in X, x \in (V - T)^*, c \in X \cup \{1\}$ or $a \in V - T, b \in Y \cup 1, x \in T^*, c \in Y$.*

Proof. See Lemma 1 in [10]. ■

Consider the left-extended queue grammar $Q = (V, T, W, F, s, R)$ from Lemma 2. Its properties imply that Q generates every word in $L(Q)$ so that it passes through state 1. Before it enters 1, it generates only words over $(V - T)$; after entering 1, it generates only words over T. In greater detail, the next corollary expresses this property, which fulfills a crucial role in the proof of Lemma 4.

Corollary 3. *Q constructed in the proof of Lemma 2 generates every $h \in L(Q)$ in this way*

$$
\begin{array}{lll}
 & \#a_0q_0 & \\
\Rightarrow & a_0\#x_0q_1 & [(a_0, q_0, z_0, q_1)] \\
\Rightarrow & a_0a_1\#x_1q_2 & [(a_1, q_1, z_1, q_2)] \\
\vdots & & \\
\Rightarrow & a_0a_1 \dots a_k\#x_kq_{k+1} & [(a_k, q_k, z_k, q_{k+1})] \\
\Rightarrow & a_0a_1 \dots a_ka_{k+1}\#x_{k+1}y_1q_{k+2} & [(a_{k+1}, q_{k+1}, y_1, q_{k+2})] \\
\vdots & & \\
\Rightarrow & a_0a_1 \dots a_ka_{k+1} \dots a_{k+m-1} & \\
 & \#x_{k+m-1}y_1 \dots y_{m-1}q_{k+m} & [(a_{k+m-1}, q_{k+m-1}, y_{m-1}, q_{k+m})] \\
\Rightarrow & a_0a_1 \dots a_ka_{k+1} \dots a_{k+m}\#y_1 \dots y_mq_{k+m+1} & [(a_{k+m}, q_{k+m}, y_m, q_{k+m+1})]
\end{array}
$$

where $k, m \geq 1, a_i \in V - T$ *for* $i = 0, \dots, k + m, x_j \in (V - T)^*$ *for* $j = 1, \dots, k + m, s = a_0q_0, a_jx_j = x_{j-1}z_j$ *for* $j = 1, \dots, k, a_1 \dots a_kx_{k+1} = z_0 \dots z_k, a_{k+1} \dots a_{k+m} = x_k, q_0, q_1, \dots, q_{k+m} \in W - F$ *and* $q_{k+m+1} \in F, z_1, \dots, z_k \in (V - T)^*, y_1, \dots, y_m \in T^*, h = y_1y_2 \dots y_{m-1}y_m$.

Lemma 4. *Let Q be a left-extended queue grammar such that* $card(\mathbf{alph}(L(Q))) \geq 2$. *Then, there exists a centralized two-way three-linear 2-PC grammar system,* $\Gamma = (\{Q_2\}, G_1, G_2)$, *such that* $L(\Gamma) = L(Q), index(G_1) = 2, index(G_2) = 3, index(\Gamma) = 3$, *q-degree*$(\Gamma) = 1$. *In addition, Γ's master,* $G_1 = (\{Q_2\} \cup N_1, T, P_1, S_1)$, *satisfies* $card(N_1) = 3$ *and q-*$P_1 = \{A \rightarrow Q_2\}$.

Proof. Let $Q = (V, T, W, F, s, R)$ be a left-extended queue grammar such that $card(\mathbf{alph}(L(Q))) \geq 2$. Assume that $\{0, 1\} \subseteq \mathbf{alph}(L(\Gamma))) \cap T$. Furthermore, without any loss of generality, assume that Q satisfies the properties described in Lemma 2 and Corollary 3. Observe that there exist a positive integer, n, and an injection, ι, from VW to $(\{0, 1\}^n - 1^n)$ so that ι remains an injection when its domain is extended to $(VW)^*$ in the standard way (after this extension, ι thus represents an injection from $(VW)^*$ to $(\{0, 1\}^n - 1^n)^*$); a proof of this observation is simple and left to the reader. Based on ι, define the substitution, ν, from V to $(\{0, 1\}^n - 1^n)$ as $\nu(a) = \{\iota(aq) : q \in W\}$ for every $a \in V$. Extend the domain of ν to V^*. Furthermore, define the substitution, μ, from W to $(\{0, 1\}^n - 1^n)$ as $\mu(q) = \{\mathbf{reversal}(\iota(aq)) : a \in V\}$ for every $q \in W$. Extend the domain of μ to W^*. Set $o = 1^n$.

Construction. Introduce the centralized two-way three-linear 2-PC grammar system, $\Gamma = (\{Q_2\}, G_1, G_2)$, where $G_1 = (Q \cup N_1, T, P_1, S_1), G_2 = (N_2, T, P_2, S_2)$, $N_1 = \{S_1, A, Y\}$, and $P_1 = \{S_1 \to oAo, S_1 \to oYo, A \to Q_2\} \cup \{A \to \mathbf{reversal}(x)Ax : x \in \iota(VW)\} \cup \{Y \to xYx : x \in \iota(VW)\}$. P_2 is constructed as follows

1. if $s = a_0q_0$, where $a_0 \in V - T$ and $q_0 \in W - F$, then add $S_2 \to Yu\,\langle q_0, 1\rangle\, tY$ to P_2, for all $u \in \nu(a_0)$ and $t \in \mu(q_0)$,
2. if $(a, q, y, p) \in R$, where $a \in V - T, p, q \in W - F$, and $y \in (V - T)^*$, then add $\langle q, 1\rangle \to u\,\langle p, 1\rangle\, t$ to P_2, for all $u \in \nu(y)$ and $t \in \mu(p)$,
3. for every $q \in W - F$, add $\langle q, 1\rangle \to o\,\langle q, 2\rangle$ to P_2,
4. if $(a, q, y, p) \in R$, where $a \in V - T, p, q \in W - F, y \in T^*$, then add $\langle q, 2\rangle \to y\,\langle p, 2\rangle\, t$ to P_2, for all $t \in \mu(p)$,
5. if $(a, q, y, p) \in R$, where $a \in V - T, q \in W - F, y \in T^*$, and $p \in F$, then add $\langle q, 2\rangle \to yo$ to P_2,
6. add $Y \to Y$ to P_2,

and N_2 contains all symbols occurring in P_2 that are not in T.

Basic Idea. Clearly, Γ's master, $G_1 = (\{Q_2\} \cup N_1, T, P_1, S_1)$, satisfies $card(N_1) = 3$ and $q\text{-}P_1 = \{A \to Q_2\}$. Every generation of $y \in L(\Gamma)$ can be expressed as follows

$$
\begin{aligned}
& (S_1, S_2) \\
\overset{d}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_0)A\beta_0 o, Y\chi_0\,\langle q_1, 1\rangle\,\mathbf{reversal}(\beta_0)Y) \\
\overset{d}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_1)A\beta_1 o, Y\chi_1\,\langle q_2, 1\rangle\,\mathbf{reversal}(\beta_1)Y) \\
& \vdots \\
\overset{d}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_k)A\beta_k o, Y\chi_k\,\langle q_{k+1}, 1\rangle\,\mathbf{reversal}(\beta_k)Y) \\
\overset{d}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_k)A\beta_k o, Y\chi_k o\,\langle q_{k+1}, 2\rangle\,\mathbf{reversal}(\beta_k)Y) \\
\overset{d}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_k)A\beta_{k+1} o, Y\chi_k o y_1\,\langle q_{k+1}, 2\rangle\,\mathbf{reversal}(\beta_{k+1})Y) \\
& \vdots \\
\overset{d}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_{k+m})Q_2\beta_{k+m} o, Y\chi_k o y_1 \dots y_m o\mathbf{reversal}(\beta_{k+m})Y) \\
\overset{q}{\Rightarrow}\ & (o\mathbf{reversal}(\alpha_{k+m})Y\alpha_{k+m} o y_1 \dots y_m o\mathbf{reversal}(\beta_{k+m})Y\beta_{k+m} o), \zeta) \\
\overset{r}{\Rightarrow}\ & (o\mathbf{prefix}(\mathbf{reversal}(\alpha_{k+m}), |\alpha_{k+m}| - n)Y\mathbf{suffix}(a_{k+m}, |a_{k+m}| - n) \\
& o y_1 \dots y_m o\mathbf{reversal}(\beta_{k+m})Y\beta_{k+m} o), \zeta) \\
& \vdots \\
\overset{r}{\Rightarrow}\ & (oYoy_1 \dots y_m oYo, \zeta) \\
\overset{r}{\Rightarrow}{}^2\ & (S_1 y_1 \dots y_m S_1, \zeta)
\end{aligned}
$$

where $k, m \geq 1$, and for all $e = 0, \dots, k + m, \alpha_e \in \nu(a_0 \dots a_e), \beta_e \in \mu(q_0 \dots q_e)$, $\alpha_e = \mathbf{reversal}(\beta_e), a_i \in V - T, q_i \in W - F, 1 \leq i \leq k + m$, for all $f = 0, \dots, k - 1$, $\chi_f \in \mathbf{prefix}(\nu(a_0 \dots a_e)) \cap \mathbf{prefix}(\chi_{f+1}), \chi_k = a_{k+m}, s = a_0q_0, y_1, \dots, y_m \in T^*$,

$\zeta = Y\chi_k o y_1 \dots y_m o \mathbf{reversal}(\beta_{k+m})Y, y = y_1, \dots, y_m$, and R contains rules $(a_0, q_0, z_0, q_1), (a_1, q_1, z_1, q_2), \dots, (a_{k+m}, q_{k+m}, y_{m-1}, q_{k+m+1})$ according to which Q can make the generation of y described in Corollary 3. As a result, *q-degree*$(\Gamma) = 1$ and $L(\Gamma) \subseteq L(Q)$. On the other hand, recall that Q generates every $y \in L(Q)$ as described in Corollary 3. Then, we can easily construct the above generation of y in Γ, so $L(Q) \subseteq L(\Gamma)$. Therefore, $L(\Gamma) = L(Q)$.

Formal Proof (Sketch). For brevity, the following rigorous proof omits some obvious details, which the reader can easily fill in.

Claim 1. *G generates every $h \in L(\Gamma)$ as follows (S_1, S_2) ${}_d\Rightarrow^*$ (uAv, y) ${}_q\Rightarrow$ (uyv, y) ${}_r\Rightarrow^*$ (h, y), where $u, v \in \{0,1\}^*, y \in \{Y\}(T \cup \{0,1\})^*\{Y\}$.*

Proof. In P_1, the right-hand side of every production contains a symbol from $Q \cup N_1$, so during any successful computation, Γ makes at least one q-step. The only production by which G_1 can cause Γ to make a q-step is $A \to q_2$. A does not occurr in N_2 at all, and after the first application of $A \to q_2$, G_1 makes reductions during which it can never obtain A in a sentential form. Thus, the first application of $A \to q_2$ is also the last application of this production. Therefore, Γ generates every $h \in L(\Gamma)$ as follows (S_1, S_2) ${}_d\Rightarrow^*$ (uAv, y) ${}_q\Rightarrow$ (uyv, y) ${}_r\Rightarrow^*$ (h, z),where $u, v \in \{0,1\}^*, y, z \in (T \cup N)^*$. If y contains a symbol from $N_2 - (T \cup \{Y\})$, G_1 can never remove them during (uyv, y) ${}_r\Rightarrow^*$ (h, z) by any rule from P_1, which leads to a contradiction that $h \neq L(\Gamma)$. Thus, $y, z \in (T \cup \{Y\})^*$. Examine P_2 to see that $y, z \in (T \cup \{Y\})^*$ implies $y = z$ and $y \in \{Y\}(T \cup \{0,1\})^*\{Y\}$. As a result, (S_1, S_2) ${}_d\Rightarrow^*$ (uAv, y) ${}_q\Rightarrow$ (uyv, y) ${}_r\Rightarrow^*$ (h, y), where $u, v \in \{0,1\}^*, y \in \{Y\}(T \cup \{0,1\})^*\{Y\}$. □

The previous claim implies *q-degree*$(\Gamma) = 1$.

Claim 2. *Let (S_1, S_2) ${}_d\Rightarrow^*$ (uAv, y) ${}_q\Rightarrow$ (uyv, y) ${}_r\Rightarrow^*$ (h, y) in Γ, where $h \in L(\Gamma), u, v \in \{0,1\}^*, y \in \{Y\}(T \cup \{0,1\})^*\{Y\}$. Then, $v =$ **reversal**(u).*

Proof. Examine 1-P_1. Observe that before the communicational step, G_1 can use only productions from $\{S_1 \to oAo\} \cup \{A \to \mathbf{reversal}(z)Az : z \in \iota(VW)\}$; therefore, $v =$ **reversal**(u). □

Claim 3. *Let (S_1, S_2) ${}_d\Rightarrow^*$ $(u A \mathbf{reversal}(u), y)$ ${}_q\Rightarrow$ $(uy\mathbf{reversal}(u), y)$ ${}_r\Rightarrow^*$ (h, y), in Γ, where $h \in L(\Gamma), u, v \in \{0,1\}^*, y \in \{Y\}(T \cup \{0,1\})^*\{Y\}$. Then, $y =$ $Y\mathbf{reversal}(u)huY$.*

Proof. Consider $(uy\mathbf{reversal}(u), y)$ ${}_r\Rightarrow^*$ (h, y). During 1-*computation*$((uy\mathbf{reversal}(u), y)$ ${}_r\Rightarrow^*$ $(h, y))$, G_1 can use only productions from $\{S_1 \to oYo\} \cup \{Y \to xYx : x \in \iota(VW)\}$. Thus, $y = Y\mathbf{reversal}(u)huY$. □

Return to the proof of the lemma. Let

$$\begin{aligned}(S_1, S_2) \; {}_d\Rightarrow^* \; & (uA\mathbf{reversal}(u), Y\mathbf{reversal}(u)huY) \\ {}_q\Rightarrow \; & (uY\mathbf{reversal}(u)huY\mathbf{reversal}(u), Y\mathbf{reversal}(u)huY) \\ {}_r\Rightarrow^* \; & (h, Y\mathbf{reversal}(u)huY)\end{aligned}$$

in Γ, where $u, v \in \{0,1\}^*$. Examine P_1 and P_2 to see that in greater detail this computation can be expressed as

$$
\begin{aligned}
& (S_1, S_2) \\
{}_d\Rightarrow\ & (\mathbf{oreversal}(\alpha_0)A\beta_0 o, Y\chi_0\,\langle q_1, 1\rangle\,\mathbf{reversal}(\beta_0)Y) \\
{}_d\Rightarrow\ & (\mathbf{oreversal}(\alpha_1)A\beta_1 o, Y\chi_1\,\langle q_2, 1\rangle\,\mathbf{reversal}(\beta_1)Y) \\
& \vdots \\
{}_d\Rightarrow\ & (\mathbf{oreversal}(\alpha_k)A\beta_k o, Y\chi_k\,\langle q_{k+1}, 1\rangle\,\mathbf{reversal}(\beta_k)Y) \\
{}_d\Rightarrow\ & (\mathbf{oreversal}(\alpha_k)A\beta_k o, Y\chi_k o\,\langle q_{k+1}, 2\rangle\,\mathbf{reversal}(\beta_k)Y) \\
{}_d\Rightarrow\ & (\mathbf{oreversal}(\alpha_k)A\beta_{k+1} o, Y\chi_k o y_1\,\langle q_{k+1}, 2\rangle\,\mathbf{reversal}(\beta_{k+1})Y) \\
& \vdots \\
{}_d\Rightarrow\ & (\mathbf{oreversal}(\alpha_{k+m})Q_2\beta_{k+m} o, Y\chi_k o y_1 \dots y_m \mathbf{oreversal}(\beta_{k+m})Y) \\
{}_q\Rightarrow\ & (\mathbf{oreversal}(\alpha_{k+m})Y\alpha_{k+m} o y_1 \dots y_m \mathbf{oreversal}(\beta_{k+m})Y\beta_{k+m} o), \zeta) \\
{}_r\Rightarrow\ & (\mathbf{oprefix}(\mathbf{reversal}(\alpha_{k+m}), |\alpha_{k+m}| - n)Y\mathbf{suffix}(a_{k+m}, |a_{k+m}| - n) \\
& o y_1 \dots y_m \mathbf{oreversal}(\beta_{k+m})Y\beta_{k+m} o), \zeta) \\
& \vdots \\
{}_r\Rightarrow\ & (oYoy_1 \dots y_m oYo, \zeta) \\
{}_r\Rightarrow^2\ & (S_1 y_1 \dots y_m S_1, \zeta)
\end{aligned}
$$

where $k, m \geq 1$, and for all $e = 0, \dots, k+m, \alpha_e \in \nu(a_0 \dots \alpha_e), \beta_e \in \mu(q_0 \dots q_e)$, $\alpha_e = \mathbf{reversal}(\beta_e), a_i \in V - T, q_i \in W - F, 1 \leq i \leq k+m$, for all $f = 0, \dots, k-1, \chi_f \in \mathbf{prefix}(\nu(a_0 \dots a_e)) \cap \mathbf{prefix}(\chi_{f+1}), \chi_k = \alpha_{k+m}, s = a_0 q_0$, $y_1, \dots, y_m \in T^*, \zeta = Y\chi_k o y_1 \dots y_m \mathbf{oreversal}(\beta_{k+m})Y, h = y_1, \dots, y_m$. Thus, $index(G_1) = 2, index(G_2) = 3$, and $index(\Gamma) = 3$. Recall that $\chi_k = a_{k+m}$. Consider the derivation part of the above computation—that is,

$$
\begin{aligned}
2\text{-}computation((S_1, S_2)\ {}_d\Rightarrow^*\ & (\mathbf{oreversal}(\alpha_{k+m})Q_2\beta_{k+m} o, Y\alpha_{k+m} o y_1 \dots \\
& y_m \mathbf{oreversal}(b_{k+m})Y))
\end{aligned}
$$

From the construction of P_2, the form of this computation implies that R contains rules $(a_0, q_0, z_0, q_1), (a_1, q_1, z_1, q_2), \dots, (a_{k+m}, q_{k+m}, y_{m-1}, q_{k+m+1})$, where $s = a_0 q_0, a_j x_j = x_{j-1} z_j$ for $j = 1, \dots, k, a_1 \dots a_k x_{k+1} = z_0 \dots z_k, a_{k+1} \dots a_{k+m} = x_k$, and $q_{k+m+1} \in F, z_1, \dots, z_k \in (V - T)^*, y_1, \dots, y_m \in T^*, h = y_1 y_2 \dots y_{m-1} y_m$. As a result,

$$
\begin{aligned}
& \#a_0 q_0 \\
\Rightarrow\ & a_0 \# x_0 q_1 && [(a_0, q_0, z_0, q_1)] \\
\Rightarrow\ & a_0 a_1 \# x_1 q_2 && [(a_1, q_1, z_1, q_2)] \\
& \vdots \\
\Rightarrow\ & a_0 a_1 \dots a_k \# x_k q_{k+1} && [(a_k, q_k, z_k, q_{k+1})] \\
\Rightarrow\ & a_0 a_1 \dots a_k a_{k+1} \# x_{k+1} y_1 q_{k+2} && [(a_{k+1}, q_{k+1}, y_1, q_{k+2})] \\
& \vdots \\
\Rightarrow\ & a_0 a_1 \dots a_k a_{k+1} a_{k+m-1} \# x_{k+m-1} y_1 \dots y_{m-1} q_{k+m} \\
& && [(a_{k+m-1}, q_{k+m-1}, y_{m-1}, q_{k+m})] \\
\Rightarrow\ & a_0 a_1 \dots a_k a_{k+1} a_{k+m} \# y_1 \dots y_m q_{k+m+1} && [(a_{k+m}, q_{k+m}, y_m, q_{k+m+1})]
\end{aligned}
$$

in Q. As $h = y_1 y_2 \dots y_{m-1} y_m, h \in L(Q)$. Thus, $L(\Gamma) \subseteq L(Q)$.

To prove $L(Q) \subseteq L(\Gamma)$, recall that Q satisfies the properties described in Lemma 2 and, therefore, generates every $h \in L(Q)$ as described in Corollary 3. Then, we can easily construct the generation of h in Γ that has the form described above; a detailed version of this construction is left to the reader. Thus, $h \in L(\Gamma)$, so $L(Q) \subseteq L(\Gamma)$.

Therefore, $L(\Gamma) = L(Q)$. Recall that we have already established that $index(G_1) = 2, index(G_2) = 3, index(\Gamma) = 3, q - degree(\Gamma) = 1, card(N_1) = 3$, $q\text{-}P_1 = \{AQ_2\}$. Thus, Lemma 4 holds. ■

Theorem 5. *Let L be a recursively enumerable language such that $card(\mathbf{alph}(L)) \geq 2$. Then, there exists a centralized two-way three-linear 2-PC grammar system, $\Gamma = (\{q_2\}, G_1, G_2)$, such that $L(\Gamma) = L, index(G_1) = 2, index(G_2) = 3, index(\Gamma) = 3, q\text{-}degree(\Gamma) = 1$, and Γ's master, $G_1 = (Q \cup N_1, T, P_1, S_1)$, satisfies $card(N_1) = 3, q\text{-}P_1 = \{A \to Q_2\}$.*

Proof. This theorem follows from Lemmas 1, 2, and 4. ■

5 Some Variants

This concluding section discusses some variants of the centralized two-way metalinear grammar systems.

Parallel variant. A parallel variant of a centralized two-way k-linear PC grammar system makes communication steps as defined in Section 4; however, during derivation and reduction steps, it allows their components to simultaneously rewrite the word at several places. More formally, let $\Gamma = (Q, G_1, \dots, G_n)$, where for all $i = 1, \dots, n, G_i = (Q \cup N_i, T, P_i, S_i)$ is a two-way k-linear PC component. As before, for $u, v \in (N_i \cup T)^*$ and $A \to x \in P_i$, write $uAv \;{}_d\Rightarrow\; uxv$ and $uxv \Rightarrow ruAv$ in G_i. Let $x_i, y_i \in (N \cup T)^*$, where $i = 1, \dots, n$, for some $n \geq 1$. If $x_i \;{}_d\Rightarrow\; y_i$ in G_i for all $i = 1, \dots, n$, write $x_1 \dots x_n \;{}_{par\text{-}d}\Rightarrow\; y_1 \dots y_n$ in Γ. If $x_i \;{}_r\Rightarrow\; y_i$ in G_i for all $i = 1, \dots, n$, write $x_1 \dots x_n \;{}_{par\text{-}r}\Rightarrow\; y_1 \dots y_n$ in Γ. To complete the definition of a parallel centralized two-way k-linear PC grammar system, modify the corresponding definition given in Section 3 by substituting ${}_{par\text{-}d}\Rightarrow$ and ${}_{par\text{-}r}\Rightarrow$ for ${}_d\Rightarrow$ and ${}_r\Rightarrow$, respectively. By ${}_{par}L(\Gamma)$, denote the language generated by a parallel two-way k-linear PC grammar system, Γ.

Theorem 6. *Let L be a recursively enumerable language such that $card(\mathbf{alph}(L)) \geq 2$. Then, there exists a parallel centralized two-way three-linear 2-PC grammar system, $\Gamma = (\{Q_2\}, G_1, G_2)$, such that ${}_{par}L(\Gamma) = L, index(G_1) = 2, index(G_2) = 3, index(\Gamma) = 3, q\text{-}degree(\Gamma) = 1$, and Γ's master, $G_1 = (Q \cup N_1, T, P_1, S_1)$, satisfies $card(N_1) = 3$ and $q\text{-}P_1 = \{A \to Q_2\}$.*

Proof. Establish this theorem by analogy with the demonstration of Theorem 5. ■

Terminating mode. The theory of grammar systems has introduced several derivation modes, such as *-mode or the maximal code for CD grammar systems, and studied the corresponding families of languages generated in these modes. In terms of the grammar systems discussed in this paper, we also suggest a new derivation mode, called the *terminating mode.* That is, for a centralized 2-PC two-way metalinear grammar system, Γ, introduced in Section 3, the *language generated by* Γ *in the terminating mode,* ${}_tL(\Gamma)$, is defined by this equivalence: $L(\Gamma)$ contains $z \in T^*$ if and only if there exists $\alpha \in \Theta$ such that Γ makes $\sigma \Rightarrow^* \alpha$ but cannot make any further computational step from α and the deletion of each S_1 in 1-α results in z.

Theorem 7. *Let L be a recursively enumerable language such that* $card(\mathbf{alph}(L))) \geq 2$. *Then, there exists a parallel centralized two-way three-linear 2-PC grammar system,* $\Gamma = (\{Q_2\}, G_1, G_2)$, *such that* ${}_tL(\Gamma) = L, index(G_1) = 2, index(G_2) = 3, index(\Gamma) = 3; q\text{-}degree(\Gamma) = 1$, *and* Γ*'s master,* $G_1 = (Q \cup N_1, T, P_1, S_1)$, *satisfies* $card(N_1) = 4$ *and* $q\text{-}P_1 = \{A \rightarrow Q_2\}$.

Proof. Return to the centralized two-way metalinear 2-PC grammar system, $\Gamma = (\{Q_2\}, G_1, G_2)$, constructed in the proof of Lemma 4. Modify its master, $G_1 = (Q \cup N_1, T, P_1, S_1)$, as follows. First, add a new nonterminal, X, to N_1. Then, include $\{X \rightarrow X\} \cup \{X \rightarrow xYy \mid x, y \in \iota(VW), x \neq y\}$ into P_1. Complete this proof by analogy with the proofs of Lemma 4 and Theorem 5. ■

Returning mode. As stated in Section 1, this paper considers only the nonreturning mode throughout. Reconsider the present study in terms of returning mode (see [7]).

Acknowledgement

The author thanks the anonymous referee for several useful comments. The author also gratefully acknowledge support of GAČR grant 201/04/0441.

References

[1] Csuhaj-Varju, E.: Cooperating Grammar Systems. Power and Parameters, *LNCS* 812, Springer, Berlin, 67-84, 1994.

[2] Csuhaj-Varju, E.: Grammar Systems: a Multi-Agent Framework for Natural Language Generation, in Gh. Paun (ed.), *Artificial Life: Grammatical Models*, The Black Sea Univer. Press, Bucharest, 1995.

[3] Csuhaj-Varju, E. and Kelemen, J.: On the Power of Cooperation: a Regular Representation of R.E. Languages, *Theor. Computer Sci.* 81, 305-310, 1991.

[4] Csuhaj-Varju, E., Dassow, J., Kelemen, J., Paun, Gh.: *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[5] Csuhaj-Varju, E., Dassow, J., Kelemen, J., Paun, Gh.: Eco-Grammar Systems: A Grammatical Framework for Life-like Interactions, *Artificial Life* 3, 27-38, 1996.

[6] Csuhaj-Varju, E. and Salomaa, A.: Networks of Language Processors: Parallel Communicating Systems, *EATCS Bulletin* 66, 122-138, 1997.

[7] Dassow, J., Paun, Gh., and Rozenberg, G.: Grammar Systems. In *Handbook of Formal Languages*, Rozenberg, G. and Salomaa, A. (eds.), Volumes 2, Springer, Berlin 1997

[8] Kleijn, H. C. M. and Rozenberg, G.: On the Generative Power of Regular Pattern Grammars, *Acta Informatica* 20, 391-411, 1983.

[9] Meduna, A.: *Automata and Languages: Theory and Applications*, Springer, London, 2000.

[10] Meduna, A.: Simultaneously One-Turn Two-Pushdown Automata, *International Journal of Computer Mathematics* 80, 679-687, 2003.

[11] Paun, Gh., Salomaa, A. and S. Vicolov, S.: On the Generative Capacity of Parallel Communicating Grammar Systems, *International Journal of Computer Mathematics* 45, 45-59, 1992.

[12] Paun, Gh. and Santean, L.: Parallel Communicating Grammar Systems: the Regular Case, *Ann. Univ. Buc., Ser. Matem.-Inform.* 38, 55-63, 1989.

[13] Paun, Gh. and Santean, L.: Further Remarks about Parallel Communicating Grammar Systems, *International Journal of Computer Mathematics* 34, 187-203, 1990.

[14] Salomaa, A.: *Formal Languages*, Academic Press, New York, 1973.

[15] Santean, L.: Parallel Communicating Systems, *EATCS Bulletin*, 160-171, 1990.

[16] Vaszil, G.: On simulating Non-returning PC grammar Systems with Returning Systems, *Theoretical Computer Science* (209) 1-2, 319-329, 1998.

Received February, 2003

Acta Cybernetica **16** (2004) 399–409.

Retractable state-finite automata without outputs*

Attila Nagy†

Abstract

A homomorphism of an automaton **A** without outputs onto a subautomaton **B** of **A** is called a retract homomorphism if it leaves the elements of B fixed. An automaton **A** is called a retractable automaton if, for every subautomaton **B** of **A**, there is a retract homomorphism of **A** onto **B**. In [1] and [3], special retractable automata are examined. The purpose of this paper is to give a construction for state-finite retractable automata without outputs.

In this paper, by an automaton we mean an automaton without outputs, that is, a system $\mathbf{A} = (A, X, \delta)$ consisting of a non-empty *state set* A, a non-empty *input set* X and a *transition function* $\delta : A \times X \mapsto A$. If A has only one element then the automaton **A** will be called *trivial*. The function δ is extended to $A \times X^*$ (X^* denotes the free monoid over X) as follows. If a is an arbitrary state of **A** then $\delta(a, e) = a$ for the empty word e, and $\delta(a, qx) = \delta(\delta(a, q), x)$ for every $q \in X^*$, $x \in X$.

If B is a non-empty subset of the state-set of an automaton $\mathbf{A} = (A, X, \delta)$ such that $\delta(b, x) \in B$ for every $b \in B$ and $x \in X$, then $\mathbf{B} = (B, X, \delta_B)$ is an automaton, where δ_B denotes the restriction of δ to $B \times X$. This automaton is called a *subautomaton* (more precisely, an *A-subautomaton*) of **A**. A subautomaton **B** of an automaton **A** is called a *proper subautomaton* of **A** if B is a proper subset of A. A subautomaton **B** of an automaton **A** is said to be a *minimal subautomaton* of **A** if **B** has no proper subautomaton. If a subautomaton **B** of an automaton **A** has only one state then **B** is minimal; the state of **B** is called a *trap* of **A**. If an automaton $\mathbf{A} = (A, X, \delta)$ contains only one trap denoted by a_0 then **A** is called a *one-trap automaton* (or an OT-*automaton*). This fact will be denoted by $(A, X, \delta; a_0)$. If an automaton **A** has a subautomaton which is contained in every subautomaton of **A** then it is called the *kernel* of **A**. The kernel of **A** is denoted by $Ker\mathbf{A}$.

Let $\mathbf{A} = (A, X, \delta)$ be an automaton containing at most one trap. Let A^0 denote the following set. $A^0 = A$ if **A** does not contain a trap or **A** is trivial; $A^0 = A - \{a_0\}$ if **A** is a non-trivial OT-automaton and a_0 is the trap of **A**. Consider the mapping $\delta^0 : A^0 \times X \mapsto A^0$ which is defined for a couple $(a, x) \in A^0 \times X$ if and only if

*Research supported by the Hungarian NFSR grant No T029525 and No T042481

†Department of Algebra, Institute of Mathematics, Budapest University of Technology and Economics

$\delta(a,x) \in A^0$. In this case, let $\delta^0(a,x) = \delta(a,x)$. (A^0, X, δ^0) is a *partial automaton* which will be denoted by $\mathbf{A}^0$.

An equivalence relation α of the state set A of an automaton $\mathbf{A} = (A, X, \delta)$ is called a *congruence* of $\mathbf{A}$ if, for every $a, b \in A$ and $x \in X$, the assumption $(a,b) \in \alpha$ implies $(\delta(a,x), \delta(b,x)) \in \alpha$. It is easy to see that if $\mathbf{B}$ is a subautomaton of an automaton $\mathbf{A}$ then $\rho_B = \{(a,b) \in A \times A : a = b \text{ or } a, b \in B\}$ is a congruence of $\mathbf{A}$, which is called the *Rees congruence* of $\mathbf{A}$ induced by $\mathbf{B}$. The factor automaton $\mathbf{A}/\rho_B$ is called the *Rees factor automaton* of $\mathbf{A}$ modulo $\mathbf{B}$. If $\mathbf{B}$ is a subautomaton of an automaton $\mathbf{A}$ then we may describe the Rees factor $\mathbf{A}/\rho_B$ as the result of collapsing B into a trap a_0 of the Rees factor, while the elements of A outside of B retain their identity. Sometimes we can identify these elements a $(a \in A - B)$ with the one-element ρ_B-class $[a]$, that is, we can suppose that the state set of the Rees factor is $(A - B) \cup \{a_0\}$.

If a is a state of an automaton $\mathbf{A}$, then the smallest subautomaton $\mathbf{R}(a)$ of $\mathbf{A}$ containing the state a is called the *principal subautomaton* of $\mathbf{A}$ generated by a. It is easy to see that $R(a) = \delta(a, X^*) = \{\delta(a,p) : p \in X^*\}$. Clearly, every minimal subautomaton of an automaton is principal.

The relation $\mathcal{R}$ on an automaton $\mathbf{A}$ defined by $\mathcal{R} = \{(a,b) \in A \times A : R(a) = R(b)\}$ is an equivalence relation on A. The $\mathcal{R}$-class of A containing an element $a \in A$ is denoted by R_a. The subset $R(a) - R_a$ is denoted by $R[a]$. It is clear that $R[a]$ is either empty or $(R[a], X, \delta_{R[a]})$ is a subautomaton of $\mathbf{A}$. The factor automaton $\mathbf{R}\{a\} = \mathbf{R}(a)/\rho_{R[a]}$ is called a *principal factor* of $\mathbf{A}$. We note that if $\mathbf{R}[a] = \emptyset$ then $\mathbf{R}\{a\}$ is defined to be $\mathbf{R}(a)$. For example, if a is a trap then $R(a) = \{a\}$ and so $R[a] = \emptyset$.

A mapping ϕ (acting on the left) of the state set A of an automaton $\mathbf{A} = (A, X, \delta_A)$ into the state set B of an automaton $\mathbf{B} = (B, X, \delta_B)$ is called a *homomorphism* of $\mathbf{A}$ into $\mathbf{B}$ if $\phi(\delta_A(a,x)) = \delta_B(\phi(a), x)$ for every $a \in A$ and $x \in X$.

A mapping ϕ (acting on the left) of A^0 into B^0 is called a *partial homomorphism* of a partial automaton $\mathbf{A}^0 = (A^0, X, \delta_A^0)$ into a partial automaton $\mathbf{B}^0 = (B^0, X, \delta_B^0)$ if, for every $a \in A^0$, $x \in X$, the assumption $\delta_A(a,x) \in A^0$ implies $\delta_B(\phi(a), x) \in B^0$ and $\delta_B(\phi(a), x) = \phi(\delta_A(a,x))$.

Definition 1. *A subautomaton* $\mathbf{B}$ *of an automaton* $\mathbf{A}$ *is said to be a retract subautomaton if there is a homomorphism of* $\mathbf{A}$ *onto* $\mathbf{B}$ *which leaves the elements of* B *fixed. Such a homomorphism is called a retract homomorphism of* $\mathbf{A}$ *onto* $\mathbf{B}$.

Definition 2. *An automaton* $\mathbf{A}$ *is called a retractable automaton if every subautomaton of* $\mathbf{A}$ *is retract.*

Lemma 1. *Every subautomaton of a retractable automaton is retractable.*

Proof. As a subautomaton $\mathbf{C}$ of a subautomaton $\mathbf{B}$ of an automaton $\mathbf{A}$ is also a subautomaton of $\mathbf{A}$, and the retriction of a retract homomorphism of $\mathbf{A}$ onto $\mathbf{C}$ to $\mathbf{B}$ is a retract homomorphism of $\mathbf{B}$ onto $\mathbf{C}$, our assertion is obvious. □

Lemma 2. *If* $\mathbf{A}$ *is a retractable automaton and* $\{a_i : i \in I\}$ *are elements of* A *such that* $R(a_i) \subseteq R(b)$ *for an element* b *of* A *then there is an index* $j \in I$ *such that* $R(a_i) \subseteq R(a_j)$ *for every* $i \in I$.

Proof. Let $\mathbf{A} = (A, X, \delta)$ be a retractable automaton and $\{a_i : i \in I\}$ be arbitrary elements of A such that $R(a_i) \subseteq R(b)$ for an element b of A. Let $R = \cup_{i \in I} R(a_i)$. As $\mathbf{R} = (R, X, \delta_R)$ is a subautomaton of $\mathbf{A}$, there is a retract homomorphism λ_R of $\mathbf{A}$ onto $\mathbf{R}$. As $\lambda_R(b) \in R$, there is an index $j \in I$ such that $\lambda_R(b) \in R(a_j)$. Then $\lambda_R(\delta(b,p)) = \delta(\lambda_R(b), p) \in R(a_j)$ for every $p \in X^*$, and so $\lambda_R(R(b)) \subseteq R(a_j)$. As $R(a_i) \subseteq R \cap R(b)$ $(i \in I)$, we get $R(a_i) = \lambda_R(R(a_i)) \subseteq R(a_j)$ for every $i \in I$. □

Corollary 1. *Every subautomaton of a principal subautomaton of a retractable automaton is principal. In particular, for every state a of a retractable automaton $\mathbf{A}$, $R[a]$ is either empty or $\mathbf{R}[a]$ is a principal subautomaton of $\mathbf{A}$.*

Proof. Let $\mathbf{B}$ be a subautomaton of a principal subautomaton $\mathbf{R}(b)$ of a retractable automaton $\mathbf{A}$. Then $R(a) \subseteq R(b)$ for every $a \in B$. By Lemma 2, there is an element $c \in B$ such that $R(a) \subseteq R(c)$ for every $a \in B$. As $B = \cup_{a \in B} R(a)$, we get $B = R(c)$. □

Let T be a set with a partial ordering $\leq$ such that every two-element subset of T has a lower bound in T and every non-empty subset of T having an upper bound in T contains a greatest element. Then T is a semilattice under multiplication $*$ by letting $a * b$ $(a, b \in T)$ be the (necessarily unique) greatest lower bound of a and b in T. A semilattice which can be constructed as above is called a *tree* ([4]).

Corollary 2. *A state-finite retractable automaton $\mathbf{A}$ contains a kernel if and only if the principal subautomata of $\mathbf{A}$ form a tree with respect to inclusion.*

Proof. Let $\mathbf{A}$ be a state-finite retractable automaton. The inclusion (the inclusion of the state-sets) is a partial ordering on the set T of all principal subautomata of $\mathbf{A}$. By Lemma 2, every non-empty subset of T having an upper bound in T contains a greatest element. As every finite tree has a least element, T (which is finite) is a tree if and only if it has a least element. As the least element of T is the kernel of $\mathbf{A}$, our proof is complete. □

Lemma 3. *Every principal subautomaton of a state-finite retractable automaton contains exactly one minimal subautomaton.*

Proof. From the finiteness of the state set, it follows that every principal subautomaton contains a minimal subautomaton. As a minimal subautomaton is a principal subautomaton, our assertion follows from Lemma 2. □

Lemma 4. *If a_1, a_2 are states of a state-finite retractable automaton $\mathbf{A} = (A, X, \delta)$ such that $B_1 \subseteq R(a_1)$, $B_2 \subseteq R(a_2)$ for distinct minimal subautomata $\mathbf{B}_1$ and $\mathbf{B}_2$ of $\mathbf{A}$ then $R(a_1) \cap R(a_2) = \emptyset$.*

Proof. If $c \in R(a_1) \cap R(a_2)$ then, by Lemma 3, there is a minimal subautomaton $\mathbf{B}$ of $\mathbf{A}$ such that $B \subseteq R(c) \subseteq R(a_1) \cap R(a_2)$. Using again Lemma 3, we get $B_1 = B = B_2$ which is a contradiction. □

If $\mathbf{A}_i = (A_i, X, \delta_i)$, $i \in I$ are automata such that $A_i \cap A_j = \emptyset$ for every $i \neq j$, then $\mathbf{A} = (A, X, \delta)$ is an automaton, where $A = \cup_{i \in I} A_i$ and $\delta(a, x) = \delta_i(a, x)$ for every $a \in A_i$ and $x \in X$. The automaton $\mathbf{A}$ is called the *direct sum* of the automata $\mathbf{A}_i$, $i \in I$.

Definition 3. *We say that an automaton* $\mathbf{A}$ *is a strong direct sum of a family of subautomata* $\mathbf{A}_i$, $i \in I$ *if* $\mathbf{A}$ *is a direct sum of* $\mathbf{A}_i$, $i \in I$ *and, for every couple* $(i, j) \in I \times I$, *there is a homomorphism of* $\mathbf{A}_i$ *into* $\mathbf{A}_j$.

Theorem 1. *A strong direct sum of retractable automata is retractable.*

Proof. Assume that an automaton $\mathbf{A} = (A, X, \delta)$ is a strong direct sum of automata $\mathbf{A}_i = (A_i, X, \delta_i)$, $i \in I$. Let $\phi_{i,j}$ be the corresponding homomorphism of $\mathbf{A}_i$ into $\mathbf{A}_j$ $(i, j \in I)$. Let $\mathbf{R}$ be an arbitrary subautomaton of $\mathbf{A}$. Let $R_i = R \cap A_i$. It is clear that R_i is either empty or $\mathbf{R}_i = (R_i, X, \delta_{R_i})$ is a subautomaton of $\mathbf{A}_i$. Let λ_{R_i} denote a retract homomorphism of $\mathbf{A}_i$ onto $\mathbf{R}_i$ if $R_i \neq \emptyset$, and let i_0 denote a fixed index, for which $R_{i_0} \neq \emptyset$. We define a mapping λ_R of A onto R as follows. If $a \in A_i$ and $R_i = \emptyset$, then let $\lambda_R(a) = \lambda_{R_{i_0}}(\phi_{i,i_0}(a))$; if $a \in A_i$ and $R_i \neq \emptyset$, then let $\lambda_R(a) = \lambda_{R_i}(a)$. It is clear that λ_R mapps A onto R and leaves the elements of R fixed. To prove that λ_R is a homomorphism of $\mathbf{A}$ onto $\mathbf{R}$, let $i \in I$, $a \in A_i$, $x \in X$ be arbitrary elements. In case $R_i = \emptyset$,

$$\lambda_R(\delta(a, x)) = \lambda_{R_{i_0}}(\phi_{i,i_0}(\delta_i(a, x))) = \lambda_{R_{i_0}}(\delta_{i_0}(\phi_{i,i_0}(a), x)) =$$

$$= \delta_{i_0}(\lambda_{R_{i_0}}(\phi_{i,i_0}(a)), x) = \delta(\lambda_R(a), x),$$

and, in case $R_i \neq \emptyset$,

$$\lambda_R(\delta(a, x)) = \lambda_{R_i}(\delta_i(a, x)) = \delta_i(\lambda_{R_i}(a), x) = \delta(\lambda_R(a), x),$$

because $a, \delta(a, x) \in A_i$. Hence λ_R is a retract homomorphism of $\mathbf{A}$ onto $\mathbf{R}$. Thus the theorem is proved. □

Theorem 2. *For a state-finite automaton* $\mathbf{A} = (A, X, \delta)$, *the following assertions are equivalent:*

(i) $\mathbf{A}$ *is retractable;*

(ii) $\mathbf{A}$ *is a direct sum of finite many state-finite retractable automata containing kernels being isomorphic to each other.*

(iii) $\mathbf{A}$ *is a strong direct sum of finite many state-finite retractable automata containing kernels.*

Proof. (i) implies (ii): Assume that $\mathbf{A}$ is retractable. As $\mathbf{A}$ is finite, it has a minimal subautomaton. Let $\{\mathbf{B}_i, i = 1, 2, \ldots r\}$ be the set of all distinct minimal subautomata of $\mathbf{A}$. Let $A_i = \cup_{a \in A}\{R(a) : B_i \subseteq R(a)\}$, $i = 1, 2, \ldots, r$. It is clear that $\mathbf{A}_i$ is a subautomaton of $\mathbf{A}$ and $\mathbf{B}_i$ is the kernel of $\mathbf{A}_i$ for every $i = 1, \ldots, r$. By Lemma 3, for every principal subautomaton $\mathbf{R}(a)$ of $\mathbf{A}$, there is a unique index i such that $B_i \subseteq R(a)$. Thus $A = \cup_{i=1}^{r} A_i$. By Lemma 4, $A_i \cap A_j = \emptyset$ for every

$i \neq j$. Hence $\mathbf{A}$ is a direct sum of the automata $\mathbf{A}_i$, $i = 1, \ldots, r$. By Lemma 1, every automaton $\mathbf{A}_i$ is retractable. Let $i, j \in \{1, 2, \ldots, r\}$ be arbitrary. As $\mathbf{B}_i$ is a minimal subautomaton of $\mathbf{A}$, the retract homomorphism λ_{B_i} of $\mathbf{A}$ onto $\mathbf{B}_i$ maps $\mathbf{B}_j$ onto $\mathbf{B}_i$. Thus $|B_j| \geq |B_i|$. Similarly, $|B_i| \geq |B_j|$. Thus $|B_i| = |B_j|$ and the restriction of λ_{B_j} to B_i is an isomorphism of $\mathbf{B}_i$ onto $\mathbf{B}_j$. Thus (ii) is satisfied.

(ii) implies (iii): Assume that $\mathbf{A}$ is a direct sum of the state-finite retractable automata $\mathbf{A}_i$, $i = 1, 2, \ldots, r$ such that each of $\mathbf{A}_i$ contains a kernel $\mathbf{B}_i$, and, for every $i, j \in \{1, 2, \ldots, r\}$, there is an isomorphism $\phi_{i,j}$ of $\mathbf{B}_i$ onto $\mathbf{B}_j$. It is easy to see that $\Phi_{i,j}$ defined by

$$\Phi_{i,j}(a) = \phi_{i,j}(\lambda_{B_i}(a)), \ a \in A_i$$

is a homomorphism of $\mathbf{A}_i$ into $\mathbf{A}_j$, where λ_{B_i} denotes a retract homomorphism of $\mathbf{A}_i$ onto $\mathbf{B}_i$. Thus $\mathbf{A}$ satisfies (iii).

(iii) implies (i): By Theorem 1, it is obvious. □

By the previous theorem, we concentrate our attention to state-finite retractable automata containing a kernel. These automata will be described by Corollary 3 and Theorem 7. First consider some results and notions which will be needed for us.

Lemma 5. *Every principal factor of an automaton can contain at most one trap.*

Proof. If $R[a] = \emptyset$ for a state a then the principal factor $\mathbf{R}\{a\}$ has a trap only that case when a is a trap of $\mathbf{A}$, that is, the principal factor is trivial. If $R[a] \neq \emptyset$ then $R(b) = R(a)$ for every $b \in R_a = R(a) - R[a]$, and so $\mathbf{R}\{a\}$ contains only one trap, namely the $\rho_{R[a]}$-class $R[a]$ of $\mathbf{R}(a)$. □

Definition 4. *An automaton $\mathbf{A} = (A, X, \delta)$ is called strongly connected if, for every couple $(a, b) \in A \times A$, there is a word $p \in X^+$ (X^+ denotes the free semigroup over X) such that $b = \delta(a, p)$.*

We note that every strongly connected automaton can contain only one subautomaton, namely itself. We also note that if an automaton is trivial (has only one state which is a trap) then it is strongly connected. If an automaton has at least two state and has a trap then it is not strongly connected.

Definition 5. *A non-trivial OT-automaton $\mathbf{A} = (A, X, \delta; a_0)$ is called strongly trap-connected if, for every couple $(a, b) \in A \times A$, $a \neq a_0$, there is a word $p \in X^+$ such that $b = \delta(a, p)$.*

We note that every strongly trap-connected automaton $\mathbf{A} = (A, X, \delta; a_0)$ contains only two subautomaton, namely itself and $(\{a_0\}, X, \delta_{\{a_0\}})$. Moreover, for every state $a \neq a_0$ of $\mathbf{A}$ there is a word $p \in X^+$ such that $a = \delta(a, p)$.

Definition 6. *We say that a non-trivial OT-automaton $\mathbf{A} = (A, X, \delta; a_0)$ is strongly trapped if $\delta(a, x) = a_0$ for every $a \in A$ and $x \in X$.*

Theorem 3. *Every principal factor of an automaton is either strongly connected or strongly trap-connected or strongly trapped.*

Proof. If $R[a] = \emptyset$ then $\mathbf{R}\{a\} = \mathbf{R}(a)$ is strongly connected. If $R[a] \neq \emptyset$ then, by Lemma 5, $\mathbf{R}\{a\}$ is a non-trivial OT-automaton. Let a_0 denote the trap of $\mathbf{R}\{a\}$. If $|R_a| = 1$, that is, $R\{a\} = \{a, a_0\}$, then $\mathbf{R}\{a\}$ is either strongly trapped (if $\delta(a,x) \in R[a]$ in $\mathbf{A}$, that is, $\delta(a,x) = a_0$ in $\mathbf{R}\{a\}$ for every $x \in X$) or strongly trap-connected (if $a = \delta(a,x)$ for some $x \in X$). If $|R_a| > 1$ then, for every elements b, c of R_a, $c = \delta(b,p)$ for some $p \in X^+$. Moreover, for every $b \in R_a$, there is a word $p \in X^+$ such that $\delta(b,p) \in R[a]$ in $\mathbf{A}$, that is, $\delta(b,p) = a_0$ in $\mathbf{R}\{a\}$. Hence $\mathbf{R}\{a\}$ is strongly trap-connected. □

Definition 7. *An automaton* $\mathbf{A}$ *is called semiconnected if every principal factor of* $\mathbf{A}$ *is either strongly connected or strongly trap-connected.*

Theorem 4. *An automaton* $\mathbf{A} = (A, X, \delta)$ *is semiconnected if and only if every subautomaton* $\mathbf{B}$ *of* $\mathbf{A}$ *satisfies the following: for every* $a \in B$ *there are elements* $b \in B$ *and* $p \in X^+$ *such that* $a = \delta(b,p)$.

Proof. Let $\mathbf{A} = (A, X, \delta)$ be a semiconnected automaton and $\mathbf{B}$ be a subautomaton of $\mathbf{A}$. Let a be an arbitrary element of B. Then $R(a) \subseteq B$. If a is a trap then $a = \delta(a,x)$ for every $x \in X$. Consider the case when a is not a trap. Then $|R(a)| \geq 2$. If $R[a] = \emptyset$ then, by Theorem 3, $\mathbf{R}(a) = \mathbf{R}\{a\}$ is strongly connected which means that, for every $b \in R(a)$ there is a word $p \in X^+$ such that $a = \delta(b,p)$. If $R[a] \neq \emptyset$ then, by Theorem 3, $\mathbf{R}\{a\}$ is strongly trap-connected and so, for every element $b \in R_a$, there is a word $p \in X^+$ such that $a = \delta(b,p)$. Thus, in all cases, there is a state $b \in B$ and a word $p \in X^+$ such that $a = \delta(b,p)$.

Conversely, assume that every subautomaton of an automaton $\mathbf{A}$ satisfies the condition of the theorem. We show that $\mathbf{A}$ is semiconnected. Let a be an arbitrary element of A. If a is a trap of $\mathbf{A}$ then the principal factor $\mathbf{R}\{a\}$ is trivial (and so it is strongly connected). Consider the case when a is not a trap of $\mathbf{A}$. Then a is an element of $\mathbf{R}\{a\}$ (and is not the trap of $\mathbf{R}\{a\}$). By Theorem 3, it is sufficient to show that the principal factor $\mathbf{R}\{a\}$ is not strongly trapped. As $\mathbf{R}(a)$ is a subautomaton of $\mathbf{A}$, by the condition of the theorem, there are elements $b \in R(a)$ $p \in X^*$ and $x \in X$ such that $a = \delta(b,px) = \delta(\delta(b,p),x)$ in $\mathbf{A}$. It is clear that $b' = \delta(b,p) \notin R[a]$ and so $a = \delta(b',x)$ in $\mathbf{R}\{a\}$. Thus $\mathbf{R}\{a\}$ is not strongly trapped. □

Definition 8. *Let* $\mathbf{B} = (B, X, \delta_B)$ *be a subautomaton of an automaton* $\mathbf{A} = (A, X, \delta)$. *We say that* $\mathbf{A}$ *is a dilation of* $\mathbf{B}$ *if there is a mapping* ϕ *of* A *onto* B *which leaves the elements of* B *fixed and* $\delta(a,x) = \delta_B(\phi(a),x)$ *for all* $a \in A$ *and* $x \in X$.

Theorem 5. *Every dilation of a retractable automaton is retractable.*

Proof. Let $\mathbf{A} = (A, X, \delta)$ be a dilation of a retractable subautomaton $\mathbf{B} = (B, X, \delta_B)$. Then there is a mapping ϕ of A onto B which leaves the elements of B fixed and $\delta(a,x) = \delta_B(\phi(a),x)$ for every $a \in A$ and $x \in X$. Let $\mathbf{R}$ be a subautomaton of $\mathbf{A}$. Then, for every $c \in R$ and $x \in X$, $\delta(c,x) \in R \cap B$. Let $\lambda_{R \cap B}$ denote

a retract homomorphism of $\mathbf{B}$ onto the subautomaton $\mathbf{R} \cap \mathbf{B}$. Define a mapping λ_R of A onto R as follows. Let $\lambda_R(a) = a$ if $a \in R$, and let $\lambda_R(a) = \lambda_{R \cap B}(\phi(a))$ if $a \notin R$. We show that λ_R is a homomorpism of $\mathbf{A}$ onto $\mathbf{R}$. Let $a \in A$ and $x \in X$ be arbitrary elements. If $a \in R$ then

$$\delta(\lambda_R(a), x) = \delta(a, x) = \lambda_R(\delta(a, x)).$$

Assume $a \notin R$. Then

$$\delta(\lambda_R(a), x) = \delta_B(\lambda_{R \cap B}(\phi(a)), x) =$$

$$= \lambda_{R \cap B}(\delta_B(\phi(a), x)) = \lambda_R(\delta(a, x)),$$

because $\lambda_R(a), \delta(a, x) \in B$ and the restriction of λ_R to B equals $\lambda_{R \cap B}$. Hence λ_R is a homomorphism of $\mathbf{A}$ onto $\mathbf{R}$. As λ_R leaves the elements of R fixed, it is a retract homomorphism of $\mathbf{A}$ onto $\mathbf{R}$. Consequently, $\mathbf{A}$ is a retractable automaton. □

Theorem 6. *Every retractable automaton is a dilation of a semiconnected retractable automaton.*

Proof. Let $\mathbf{A} = (A, X, \delta)$ be a retractable automaton and let $B = \delta(A, X)$. Then $\mathbf{B} = (B, X, \delta_B)$ is a subautomaton of $\mathbf{A}$ and so there is a retract homomorphism ϕ of $\mathbf{A}$ onto $\mathbf{B}$. Let $a \in A$, $x \in X$ be arbitrary elements. Then $\delta(a, x) = \phi(\delta(a, x)) = \delta_B(\phi(a), x)$. Hence $\mathbf{A}$ is a dilation of $\mathbf{B}$. By Lemma 1, $\mathbf{B}$ is retractable. Let $\mathbf{R}$ be an arbitrary subautomaton of $\mathbf{B}$. If $c \in R$ is an arbitrary element, then $c = \delta(a, x)$ for some $a \in A$ and $x \in X$. Let λ_R denote the retract homomorphism of $\mathbf{A}$ onto $\mathbf{R}$. Then $\lambda_R(a) \in R$ and

$$c = \lambda_R(c) = \lambda_R(\delta(a, x)) = \delta(\lambda_R(a), x).$$

Thus, by Theorem 4, $\mathbf{B}$ is semiconnected. □

Corollary 3. *An automaton is retractable if and only if it is a dilation of a semiconnected retractable automaton.*

Proof. By the previous two theorems, it is evident. □

Theorem 2 shows that the state-finite retractable automata are exactly the direct sums of finite many state-finite retractable automata such that each component in a mentioned direct sum contains a kernel, and these kernels are isomorphic with each other. Corollary 3 and the remark after Theorem 2 show that every component in a direct sum is a dilation of a state-finite semiconnected retractable automaton containing a kernel. Theorem 7 will show how we can construct the state-finite semiconnected retractable automata containing a kernel. These results togethet give a complete description of state-finite retractable automata.

Construction. Let T be a finite tree (under partial ordering $\leq$) with the least element i_0. Let $i \succ j$ $(i, j \in T)$ denote the fact that $i > j$ and, for every $k \in T$, $i \geq k \geq j$ implies $i = k$ or $j = k$.

Let $\mathbf{A}_i = (A_i, X, \delta_i)$, $i \in T$ be a family of disjunct automata such that

(i) $\mathbf{A}_{i_0}$ is strongly connected and $\mathbf{A}_i$ is a strongly trap-connected OT-automaton for every $i \in T$ with $i \neq i_0$.

(ii) Let $\phi_{i,i}$ denote the identity mapping of $\mathbf{A}_i$, and assume that, for every $i, j \in T$ with $i \succ j$, there is a partial homomorphism $\phi_{i,j}$ of $\mathbf{A}_i^0$ into $\mathbf{A}_j^0$ such that

(iii) for every $i \succ j$ there are elements $a \in A_i^0$ and $x \in X$ such that $\delta_i(a,x) \notin A_i^0$ and $\delta_j(\phi_{i,j}(a), x) \in A_j^0$.

For arbitrary elements $i, j \in T$ with $i \geq j$, define a partial homomorphism $\Phi_{i,j}$ of $\mathbf{A}_i^0$ into $\mathbf{A}_j^0$ as follows. $\Phi_{i,i} = \phi_{i,i}$ and, if $i > j$ such that $i \succ k_1 \succ \ldots k_n \succ j$ then let

$$\Phi_{i,j} = \phi_{k_n,j} \circ \phi_{k_{n-1},k_n} \circ \ldots \circ \phi_{k_1,k_2} \circ \phi_{i,k_1}.$$

(We note that if $i \geq j \geq k$ are arbitrary elements of T then $\Phi_{i,k} = \Phi_{j,k} \circ \Phi_{i,j}$.)

Let $A = \cup_{i \in T} A_i^0$. Define a transition function $\delta' : A \times X \mapsto A$ as follows. If $a \in A_i^0$ and $x \in X$ then let $\delta'(a,x) = \delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x)$, where $i'[a,x]$ denotes the greatest element of the set $\{j \in T : \ \delta_j(\Phi_{i,j}(a), x) \in A_j^0\}$.

It is easy to see that $\mathbf{A} = (A, X, \delta')$ is an automaton which will be denoted by $(A_i, X, \delta_i; \phi_{i,j}, T)$.

Theorem 7. *A finite automaton is a semiconnected retractable automaton containing a kernel if and only if it is isomorphic to an automaton $(A_i, X, \delta_i; \phi_{i,j}, T)$ constructed as above.*

Proof. Let $\mathbf{R}$ be a subautomaton of an automaton $(A_i, X, \delta_i; \phi_{i,j}, T)$. As every automaton $\mathbf{A}_i$ $(i \in T - \{i_0\})$ is strongly trap-connected and $\mathbf{A}_{i_0}$ is strongly connected, it follows that $R = \cup_{j \in \Gamma} A_j^0$ for some non-empty subset Γ of T. We show that Γ is an ideal of T, that is, $i \in \Gamma$ and $j \leq i$ together imply $j \in \Gamma$ for all $i, j \in T$. Let i be an arbitrary element of T such that $i \in \Gamma$, $i \neq i_0$. If $j \in T$ with $i \succ j$ then, by (iii), there are elements $a \in A_i^0$ and $x \in X$ such that $\delta_i(a,x) \notin A_i^0$ and $\delta_j(\phi_{i,j}(a), x) \in A_j^0$. Then $\delta'(a,x) \in A_j^0$. Hence $A_j^0 \cap R \neq \emptyset$ which implies that $A_j^0 \subseteq R$ and so $j \in \Gamma$. This implies that Γ is an ideal of T. As T is a tree,

$$\pi : \ i \mapsto \ \max\{\gamma \in \Gamma : \ \gamma \leq i\}$$

is a well-defined mapping of T onto Γ which leaves the elements of Γ fixed (in fact, π is a retract homomorphism of the semigroup T onto the ideal Γ of T (see [4])). We define a retract homomorphism λ_R of $\mathbf{A}$ onto $\mathbf{R}$. For an arbitrary element $a \in A$, let

$$\lambda_R(a) = \Phi_{i,\pi(i)}(a)$$

if $a \in A_i^0$. It is easy to see that λ_R leaves the elements of R fixed. We prove that λ_R is a homomorphism of $\mathbf{A}$ onto $\mathbf{R}$. Let $x \in X$, $a \in A_i^0$ be arbitrary elements. Using $\delta'(a,x) = \delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x) \in A_{i'[a,x]}^0$ and the fact that $\Phi_{i'[a,x],\pi(i'[a,x])}$ is a partial homomorphism, we get

$$\lambda_R(\delta'(a,x)) = \lambda_R(\delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x)) =$$

$$= \Phi_{i'[a,x],\pi(i'[a,x])}(\delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x)) =$$

$$= \delta_{\pi(i'[a,x])}(\Phi_{i,\pi(i'[a,x])}(a), x) \in A^0_{\pi(i'[a,x])}.$$

Using $\Phi_{i,\pi(i)}(a) \in A^0_{\pi(i)}$, we have

$$\delta'(\lambda_R(a), x) = \delta'(\Phi_{i,\pi(i)}(a), x) =$$

$$= \delta_{(\pi(i))'[\Phi_{i,\pi(i)}(a),x]}(\Phi_{\pi(i),(\pi(i))'[\Phi_{i,\pi(i)}(a),x]}(\Phi_{i,\pi(i)}(a)), x) =$$

$$= \delta_{(\pi(i))'[\Phi_{i,\pi(i)}(a),x]}(\Phi_{i,(\pi(i))'[\Phi_{i,\pi(i)}(a),x]}(a), x) \in A^0_{(\pi(i))'[\Phi_{i,\pi(i)}(a),x]}.$$

To prove that $\lambda_R(\delta'(a,x)) = \delta'(\lambda_R(a), x)$, it is sufficient to show that

$$(\pi(i))'[\Phi_{i,\pi(i)}(a), x] = \pi(i'[a,x]).$$

First, assume $i'[a,x] \geq \pi(i)$ (and so $\pi(i'[a,x]) = \pi(i)$). As $\phi_{i'[a,x],\pi(i)}$ is a partial homomorphism of $A^0_{i'[a,x]}$ into $A^0_{\pi(i)}$ and $\delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x) \in A^0_{i'[a,x]}$, we get

$$\delta_{\pi(i)}(\Phi_{i,\pi(i)}(a), x) = \delta_{\pi(i)}(\Phi_{i'[a,x],\pi(i)}(\Phi_{i,i'[a,x]}(a)), x) =$$

$$= \Phi_{i'[a,x],\pi(i)}(\delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x)) \in A^0_{\pi(i)}$$

and so

$$(\pi(i))'[\Phi_{i,\pi(i)}(a), x] = \pi(i) = \pi(i'[a,x]).$$

Next, consider the case when $i'[a,x] < \pi(i)$ (and so $\pi(i'[a,x]) = i'[a,x]$). If $j \in T$ with $\pi(i) \geq j > i'[a,x]$ then we have

$$\delta_j(\Phi_{\pi(i),j}(\Phi_{i,\pi(i)}(a)), x) = \delta_j(\Phi_{i,j}(a), x) \notin A^0_j.$$

Then

$$(\pi(i))'[\Phi_{i,\pi(i)}(a), x] \leq i'[a,x].$$

As

$$\delta_{i'[a,x]}(\Phi_{\pi(i),i'[a,x]}(\Phi_{i,\pi(i)}(a)), x) = \delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a), x) \in A^0_{i'[a,x]},$$

we get

$$(\pi(i))'[\Phi_{i,\pi(i)}(a), x] \geq i'[a,x].$$

Hence

$$(\pi(i))'[\Phi_{i,\pi(i)}(a), x] = i'[a,x] = \pi(i'[a,x]).$$

Consequently, $(\pi(i))'[\Phi_{i,\pi(i)}(a), x] = \pi(i'[a,x])$ in both cases. Hence λ_R is a (retract) homomorphism of $\mathbf{A}$ onto $\mathbf{R}$. Thus $\mathbf{A} = (A_i, X, \delta_i; \phi_{i,j}, T)$ is a retractable automaton.

We show that $\mathbf{A}$ is semiconnected. If $\mathbf{R}$ is an arbitrary subautomaton of $\mathbf{A}$, then there is an ideal Γ of T such that $R = \cup_{j \in \Gamma} A^0_j$ (see above). Let $a \in R$ be an arbitrary element. Then $a \in A^0_k$ for some $k \in \Gamma$. As A_k is strongly connected or strongly trap-connected, there are elements $b \in A^0_k$ and $p \in X^+$ such that $a = \delta_k(b,p) = \delta'(b,p)$. By Theorem 4, it means that $\mathbf{A}$ is semiconnected. As i_0 is contained in every ideal of T, $\mathbf{A}_{i_0}$ is the kernel of $(A_i, X, \delta_i; \phi_{i,j}, T)$.

Conversely, let $\mathbf{A}$ be a finite semiconnected retractable automaton containing a kernel. Let $Prf(A)$ denote the set of all principal factors of $\mathbf{A}$. By Corollary 2, $Prf(A)$ is a (finite) tree under partial ordering $\leq$ defined by $\mathbf{R}\{a\} \leq \mathbf{R}\{b\}$ if and only if $R(a) \subseteq R(b)$. As $\mathbf{A}$ is semiconnected, the least element of $Prf(A)$ is strongly connected, the other ones are strongly trap-connected.

Let T be a set with $|T| = |Prf(A)|$. Denote a bijection of T onto $Prf(A)$ by f. Define a partial ordering $\leq$ on T by $i \leq j$ $(i, j \in T)$ if and only if $f(i) \leq f(j)$. Let i_0 denote the least element of T. Clearly, T is a finite tree with the least element i_0. For every element $i \in T$, fix an element a_i in A such that $f(i) = \mathbf{R}\{a_i\}$. (We note that $\mathbf{R}\{a_i\} = \mathbf{R}\{a_j\}$ iff $a_i = a_j$ iff $i = j$). As $\mathbf{R}\{a_{i_0}\}$ is strongly connected and $\mathbf{R}\{a_i\}$ is strongly trap-connected if $i \neq i_0$, condition (i) of the Construction is satisfied.

Let $\lambda_{R(a_j)}$ $(j \in T)$ denote a fix retract homomorphism of $\mathbf{A}$ onto $\mathbf{R}(a_j)$. For every $i, j \in T$ with $i \succeq j$, let $\lambda_{i,j}$ denote the restriction of $\lambda_{R(a_j)}$ to $R(a_i)$. It is obvious that $\lambda_{i,j}$ is a retract homomorphism of $\mathbf{R}(a_i)$ onto $\mathbf{R}(a_j)$ for every $i \succeq j$, $(i, j \in T)$. Moreover, $\lambda_{i,i}$ is the identity mapping of $\mathbf{R}(a_i)$, for every $i \in T$. We show that $\lambda_{i,j}$ maps R_{a_i} into R_{a_j}. Let $a \in R_{a_i}$ be an arbitrary element (so $R(a) = R(a_i)$). Then, for every $p \in X^*$, $\lambda_{i,j}(\delta(a,p)) = \delta(\lambda_{i,j}(a), p)$. If $\lambda_{i,j}(a)$ was in $R[a_j]$ then we would have $\lambda_{i,j}(\delta(a,p)) \in R[a_j]$ for every $p \in X^*$, because $\mathbf{R}[a_j]$ is a subautomaton of $\mathbf{A}$. This would imply that $\lambda_{i,j}(R(a_i)) \subseteq R[a_j]$ which is impossible, because $\lambda_{i,j}$ maps $R(a_i)$ onto $R(a_j) = R_{a_j} \cup R[a_j] \supset R[a_j]$. Hence $\lambda_{i,j}$ maps R_{a_i} into R_{a_j} and so $\lambda_{i,j}$ can be considered as a mapping of $R^0\{a_i\}$ into $R^0\{a_j\}$. If $\delta(a,x) \in R_{a_i}$ for some $a \in R_{a_i}$ and $x \in X$ then $\delta(\lambda_{i,j}(a), x) = \lambda_{i,j}(\delta(a,x)) \in R_{a_j}$. Hence $\lambda_{i,j}$ is a partial homomorphism of the partial automaton $\mathbf{R}^0\{a_i\}$ into the partial automaton $\mathbf{R}^0\{a_j\}$. Thus condition (ii) of the Construction is satisfied (for $\mathbf{A}_i = \mathbf{R}\{a_i\}$, $\phi_{i,j} = \lambda_{i,j}$).

Assume $i \succ j$. Let $b \in R_{a_j}$ be an arbitrary element. Then $a_i \neq b \in R(a_i)$ and so there is a word $p = x_1 x_2 \ldots x_n \in X^+$ $(x_1, x_2, \ldots x_n \in X)$ such that $b = \delta(a_i, p)$. Let m be the least index such that $\delta(a_i, x_1 \ldots x_m) \in R_{a_j}$. Consider an element a of R_{a_i} (or of $\mathbf{R}^0\{a_i\}$) as follows. Let $a = a_i$ if $m = 1$. Let $a = \delta(a_i, x_1 \ldots x_{m-1})$ if $m > 1$. Then $\delta(a, x_m) \notin R_{a_i}$ (or $\delta(a, x_m) \notin \mathbf{R}^0\{a_i\}$). On the other hand,

$$\delta(\lambda_{i,j}(a), x_m) = \lambda_{i,j}(\delta(a, x_m)) = \delta(a, x_m) \in R_{a_j} = R^0\{a_j\},$$

because $\lambda_{i,j}$ leaves the elements of $R(a_j)$ fixed. Thus (iii) of the Construction is satisfied (for $\phi_{i,j} = \lambda_{i,j}$, $x = x_m$).

For arbitrary elements $i, j \in T$ with $i \geq j$, define the mapping $\Phi_{i,j}$ as follows. Let $\Phi_{i,i} = \lambda_{i,i}$ and, if $i > j$ with $i \succ k_1 \succ k_2 \succ \ldots k_n \succ j$ then let

$$\Phi_{i,j} = \lambda_{k_n,j} \circ \ldots \circ \lambda_{i,k_1}.$$

It is clear that $\Phi_{i,j}$ is a retract homomorphism of $\mathbf{R}(a_i)$ onto $\mathbf{R}(a_j)$ such that it maps R_{a_i} into R_{a_j}. Thus $\Phi_{i,j}$ can be considered as a partial homomorphism of $\mathbf{R}^0\{a_i\}$ into $\mathbf{R}^0\{a_j\}$. Moreover, $\Phi_{i,k} = \Phi_{j,k} \circ \Phi_{i,j}$ for every $i, j, k \in T$ with $i \geq j \geq k$.

Construct the automaton $\mathbf{R} = (R\{a_i\}, X, \delta_i; \lambda_{i,j}, T)$, where δ_i is the transitive function of the factor automaton $\mathbf{R}\{a_i\}$ induced by δ. It is clear that the state sets

of the automata $\mathbf{R}$ and $\mathbf{A}$ are the same. We show that the transitive functions δ of $\mathbf{A}$ equals the transitive function δ' of $\mathbf{R}$. Let $i \in T$, $a \in R_{a_i} = R^0\{a_i\}$, $x \in X$ be arbitrary elements. Assume $\delta(a,x) \in R_{a_j}$ $(i \geq j)$. Let $k \in T$ with $i \geq k > j$. Then $\delta(a,x) \in R[a_k] \subset R(a_k)$ and so

$$\delta(\Phi_{i,k}(a),x) = \Phi_{i,k}(\delta(a,x)) = \delta(a,x) \notin R_{a_k} = R^0\{a_k\},$$

because $\Phi_{i,k}$ leaves the elements of $R(a_k)$ fixed. If $j \geq k$ then

$$\delta(\Phi_{i,k}(a),x) = \Phi_{i,k}(\delta(a,x)) =$$

$$= \Phi_{j,k} \circ \Phi_{i,j}(\delta(a,x)) = \Phi_{j,k}(\delta(a,x)) \in R_{a_k} = R^0\{a_k\},$$

because $\Phi_{i,j}$ leaves the element $\delta(a,x) \in R_{a_j} = R^0\{a_j\}$ fixed, and $\Phi_{j,k}$ maps R_{a_j} into R_{a_k}. Consequently $i'[a,x] = j$. Hence

$$\delta(a,x) = \Phi_{i,j}(\delta(a,x)) = \delta(\Phi_{i,j}(a),x) = \delta_j(\Phi_{i,j}(a),x) =$$

$$= \delta_{i'[a,x]}(\Phi_{i,i'[a,x]}(a),x) = \delta'(a,x).$$

Thus the theorem is proved. □

References

[1] Babcsányi, I. and A. Nagy, *Boolean-type retractable automata*, Publicationes Mathematicae, Debrecen, 48/3-4 (1996), 193-200.

[2] Gécseg, F. and I. Peák, *Algebraic Theory of Automata*, Akadémiai Kiadó, Budapest, 1972.

[3] Nagy, A., *Boolean-type retractable automata with traps*, Acta Cybernetica, Szeged, Tom. 10, Fasc. 1-2 (1991), 53-64.

[4] Tully, E.J., *Semigroups in which each ideal is a retract*, J. Austral Math. Soc., 9 (1969), 239-245.

Received February, 2003

Acta Cybernetica **16** (2004) 411–425.

Relationally defined clones of tree functions closed under selection or primitive recursion

Reinhard Pöschel*, Alexander Semigrodskikh†, and Heiko Vogler‡

Abstract

We investigate classes of tree functions which are closed under composition and primitive recursion or selection (a restricted form of recursion). The main result is the characterization of those finitary relations ϱ (on the set of all trees of a fixed signature) for which the clone of tree functions preserving ϱ is closed under selection. Moreover, it turns out that such clones are closed also under primitive recursion.

Introduction

Classes of tree functions and primitive recursion for such functions were investigated, e.g., in [FülHVV93], [EngV91], [Hup78], [Kla84]. In this paper a tree function will be an operation $f : T^n \to T$ on the set T of all trees of a given finite signature (in general one allows trees of different signature).

If a class of operations is a clone (i.e. if it contains all projections and is closed with respect to composition), then it can be described by invariant relations (cf. e.g. [PösK79], [Pös80], [Pös01]).

In this paper we apply such results from clone theory and ask which finitary relations characterize clones of tree functions that in addition are closed under primitive recursion. The answer is given in Theorem 2.1 and shows that such relations are easy to describe: they are direct products of order ideals of trees (an order ideal contains with a tree also all its subtrees). Moreover it turns out that for such clones the closure under primitive recursion is equivalent to a much weaker closure (the so-called selection or S-closure, cf. 1.5) for which no real recursion is necessary.

*Institute of Algebra, Dresden University of Technology (Germany), poeschel@math.tu-dresden.de

†Department of Algebra and Discrete Math., Ural State University, Ekaterinburg (Russia), Alexander.Semigrodskikh@usu.ru

supported by DAAD (Deutscher Akademischer Austauschdienst)

‡Institute of Theoretical Computer Science, Dresden University of Technology (Germany), vogler@inf.tu-dresden.de

1 Notions and Notation

Let $\mathbb{N} := \{0, 1, 2, \dots\}$ denote the set of natural numbers and let $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$.

1.1. Clone theoretic notions and notation. Let T be an arbitrary set (later we shall use only the set T of trees) and let $\mathrm{Op}(T)$ denote the set of all finitary operations on T, i.e. functions of the form $f : T^n \to T$ ($n \in \mathbb{N}_+$). A set $F \subseteq \mathrm{Op}(T)$ is called a *clone* if F contains all *projections* e_i^n ($n \in \mathbb{N}_+$, $i \in \{1, \dots, n\}$) defined by

$$e_i^n(x_1, \dots, x_n) = x_i \tag{1.1.1}$$

(for every $x_1, \dots, x_n \in T$) and if F is closed with respect to composition, i.e. for every n-ary $f \in F$ and m-ary $g_1, \dots, g_n \in F$ the m-ary *composition* $f(g_1, \dots, g_n)$ defined by

$$f(g_1, \dots, g_n)(x_1, \dots, x_m) := f(g_1(x_1, \dots, x_m), \dots, g_n(x_1, \dots, x_m)) \tag{1.1.2}$$

also belongs to F, $n, m \in \mathbb{N}_+$. For technical reasons 0-ary operations will not be considered in clones; they are replaced by unary constant operations, i.e., a constant $t \in T$ is replaced by the unary constant operation $T \to T : x \mapsto t$.

The least clone containing a given set F of operations over T will be denoted by $\langle F \rangle$. The projection $e_1^1 : T \to T : x \mapsto x$ is the identity mapping and will be denoted simply by e. The clone $\langle e \rangle$ generated by e is the least clone: it consists exactly of all projections.

We say that an n-ary operation $f : T^n \to T$ *preserves* an m-ary relation $\varrho \subseteq T^m$ (or, equivalently, that ϱ is *invariant* for f) if $c_1, \dots, c_n \in \varrho$ implies $f(c_1, \dots, c_n) \in \varrho$, where

$$f(c_1, \dots, c_n) := (f(c_{11}, \dots, c_{1n}), \dots, f(c_{m1}, \dots, c_{mn})) \tag{1.1.3}$$

for m-tuples $c_1 = (c_{11}, \dots, c_{m1}), \dots, c_n = (c_{1n}, \dots, c_{mn})$ (see Fig.1).

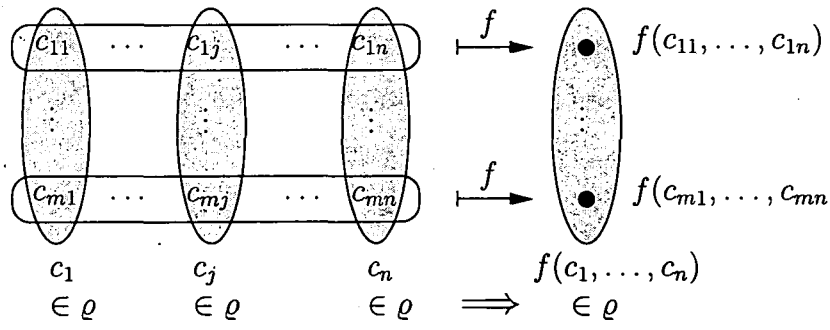

Figure 1: The operation f preserves the relation ϱ

The set of all operations preserving a given relation ϱ (so-called *polymorphisms*) will be denoted by

$$\operatorname{Pol}\varrho := \{f \mid f : T^n \to T \text{ preserves } \varrho,\ n \in \mathbb{N}_+\}\,. \tag{1.1.4}$$

It is well-known from clone theory (but also easy to see) that $\operatorname{Pol}\varrho$ is always a clone (e.g. [PösK79, 1.1.15]), moreover every clone on a finite set T can be characterized as $\operatorname{Pol}Q$ for some set Q of finitary relations ([PösK79, 1.2.1]), where

$$\operatorname{Pol}Q := \bigcap_{\varrho\in Q} \operatorname{Pol}\varrho\,.$$

For infinite T this remains true either if one considers so-called locally closed clones or if infinitary relations are allowed. We do not go into details here and refer to e.g. [Pös80], [Pös01].

We say that the i-th and j-th component of an m-ary relation $\varrho \subseteq T^m$ coincide $(i,j \in \{1,\dots,m\})$ if $a_i = a_j$ for *all* elements $(a_1,\dots,a_m) \in \varrho$. A relation ϱ is called *reduced* if no two of its components coincide. A relation can always be reduced without changing the set of polymorphisms: if the i-th and j-th component of ϱ coincide then $\operatorname{Pol}\varrho = \operatorname{Pol}\varrho'$ where $\varrho' := \{(a_1,\dots,a_{j-1},a_{j+1},\dots,a_m) \mid (a_1,\dots,a_m) \in \varrho\}$ is obtained from ϱ by deleting the j-th component. Thus we may consider reduced relations only.

1.2. Trees and subtrees. This is a well-known concept in mathematics and computer science; nevertheless we shall repeat it here in order to fix notions and notation. Let Σ be a finite *signature* (or *ranked alphabet*), i.e. a finite set of symbols such that to every symbol $\sigma \in \Sigma$ a *rank* (or *arity*) $r_\sigma \in \mathbb{N}$ is assigned. Let $\Sigma_n := \{\sigma \in \Sigma \mid r_\sigma = n\}$ denote the set of all symbols of rank n $(n \in \mathbb{N})$. We assume that $\Sigma_0 \neq \emptyset$. A *tree* (or *ground term*) over Σ is an expression which can be obtained inductively by the following rules:

(0) Every $\sigma \in \Sigma_0$ is a tree.

(1) If $\sigma \in \Sigma_r$ $(r \in \mathbb{N}_+)$ and if $s_1,\dots,s_r$ are trees, then the expression $\sigma(s_1,\dots,s_r)$ is also a tree.

If t is a tree of the form $\sigma(s_1,\dots,s_r)$ (according to (1)), then the trees $s_1,\dots,s_r$ are called *maximal subtrees* of t, and we write $s \lessdot t$ if s is a maximal subtree of t. A tree s is called *subtree* of a tree t, notation $s \leq t$, if $s = t$ or if there is a finite sequence $s_0,\dots,s_l$ of trees with

$$s = s_0 \lessdot s_1 \lessdot \dots \lessdot s_l = t$$

(i.e., $\leq$ is the transitive and reflexive closure of $\lessdot$). Note that a tree $t \in \Sigma_0$ has no proper subtrees. As usual we write $s < t$ if $s \leq t$ and $s \neq t$.

From now on T will always denote the set of all trees over a fixed signature $\Sigma = (\Sigma_r)_{r\in\mathbb{N}}$.

A subset $I \subseteq T$ of trees is an *order ideal* (*down-set*) if it is closed with respect to subtrees, i.e. if $s \leq t$ and $t \in I$, then $s \in I$.

The *height* $h(t)$ of a tree t is defined inductively on the structure of trees (according to the above rules): $h(t) := 0$ for $t \in \Sigma_0$, and $h(\sigma(s_1, \ldots, s_r)) := 1 + \max\{h(s_1), \ldots, h(s_r)\}$ for $\sigma \in \Sigma_r$ $(r \in \mathbb{N}_+)$ and $s_1, \ldots, s_r \in T$. For $k \in \mathbb{N}$ let T_k denote the set of all trees in T of height $\leq k$.

1.3. Primitive recursion for tree functions. An operation $f : T^n \to T$ will also be called *tree function*. Let $n \in \mathbb{N}_+$ and for every $\sigma \in \Sigma$ let $g_\sigma : T^{n+2r_\sigma} \to T$ be an $(n + 2r_\sigma)$-ary tree function. The $(n+1)$-ary tree function $h : T^{n+1} \to T$ defined recursively (in its last argument) by

$$h(a_1, \ldots, a_n, \sigma) := g_\sigma(a_1, \ldots, a_n) \qquad \text{for } \sigma \in \Sigma_0, a_1, \ldots, a_n \in T, \tag{1.3.1}$$

$$h(a_1, \ldots, a_n, \sigma(t_1, \ldots, t_r)) := g_\sigma(a_1, \ldots, a_n, t_1, \ldots, t_r, h(a_1, \ldots, a_n, t_1), \ldots, h(a_1, \ldots, a_n, t_r)) \tag{1.3.2}$$

$$\text{for } \sigma \in \Sigma_r,\ r \geq 1,\ \text{and } a_1, \ldots, a_n, t_1, \ldots, t_r \in T,$$

will be denoted by $\mathrm{PR}(g_\sigma)_{\sigma\in\Sigma}$; we say that h is obtained from $(g_\sigma)_{\sigma\in\Sigma}$ by *primitive recursion* (PR). By convention, equation 1.3.1 is considered as the special case of 1.3.2 for $r = 0$.

A set $F \subseteq \mathrm{Op}(T)$ of tree functions is called PR-*closed* if $\mathrm{PR}(g_\sigma)_{\sigma\in\Sigma} \in F$ whenever $g_\sigma \in F$ for all $\sigma \in \Sigma$. For $F \subseteq \mathrm{Op}(T)$, by

$$\langle\!\langle F \rangle\!\rangle_{\mathrm{PR}}$$

we shall denote the least set of tree functions which contains F and which is both, a clone and PR-closed. The existence of $\langle\!\langle F \rangle\!\rangle_{\mathrm{PR}}$ is guaranteed because the intersection of PR-closed clones is again a PR-closed clone. Obviously, $\langle F \rangle \subseteq \langle\!\langle F \rangle\!\rangle_{\mathrm{PR}}$.

1.4 Examples. a) The least PR-closed clone is $\langle\!\langle \emptyset \rangle\!\rangle_{\mathrm{PR}}$; by definition it must contain all projections and we shall denote this clone also by $\langle\!\langle e \rangle\!\rangle_{\mathrm{PR}}$. Lemma 2.4 and Proposition 2.5 shall describe some further operations which also belong to $\langle\!\langle e \rangle\!\rangle_{\mathrm{PR}}$.

b) Let F_{base} consist of all the constant tree functions $\mathrm{const}_t : T^n \to T : (t_1, \ldots, t_n) \mapsto t$ $(t \in T)$, and all the top concatenations $\mathrm{top}_\sigma : T^r \to T : (t_1, \ldots, t_r) \mapsto \sigma(t_1, \ldots, t_r)$ $(\sigma \in \Sigma_r,\ r \in \mathbb{N}_+)$. Then $\langle\!\langle F_{\mathrm{base}} \rangle\!\rangle_{\mathrm{PR}}$ is the class PREC_Σ of all primitive recursive tree functions over Σ (cf. e.g. [EngV91, 4.6]).

We are particularly interested in the following very special form of primitive recursion.

1.5. S-closure for tree functions. For a family $(g'_\sigma)_{\sigma\in\Sigma}$ of $(n + r_\sigma)$-ary tree functions g'_σ $(\sigma \in \Sigma)$ we define

$$\mathrm{S}(g'_\sigma)_{\sigma\in\Sigma} := \mathrm{PR}(g_\sigma)_{\sigma\in\Sigma} \tag{1.5.1}$$

where g_σ is the $(n+2r_\sigma)$-ary tree function defined by

$$g_\sigma := g'_\sigma(e_1^{n+2r_\sigma}, \ldots, e_{n+r_\sigma}^{n+2r_\sigma}), \tag{1.5.2}$$

for every $\sigma \in \Sigma$, i.e. we have in particular (for the $(n+1)$-ary tree function $h = \mathrm{S}(g'_\sigma)_{\sigma\in\Sigma}$)

$$\begin{aligned} h(a_1, \ldots, a_n, \sigma(t_1, \ldots, t_{r_\sigma})) = \\ g_\sigma(a_1, \ldots, a_n, t_1, \ldots, t_{r_\sigma}, h(a_1, \ldots, a_n, t_1), \ldots, h(a_1, \ldots, a_n, t_{r_\sigma})) \\ = g'_\sigma(a_1, \ldots, a_n, t_1, \ldots, t_{r_\sigma}). \end{aligned} \tag{1.5.3}$$

Analogously to the PR-closure and the notation $\langle\!\langle F \rangle\!\rangle_{\mathrm{PR}}$, we introduce the S-closure of F (using the special primitive recursion S instead of PR) and let

$$\langle\!\langle F \rangle\!\rangle_{\mathrm{S}}$$

denote the least S-closed clone containing F. We call this S-closure also *selection* closure because the functions g_σ just select and there is no real recursion (i.e. h is not allowed to call itself recursively, see 1.3.2 and 1.5.3). By definition we have

$$\langle\!\langle F \rangle\!\rangle_{\mathrm{S}} \subseteq \langle\!\langle F \rangle\!\rangle_{\mathrm{PR}} \tag{1.5.4}$$

and this inclusion is proper in general.

1.6 Remarks. a) If we write $\mathrm{PR}(g_\sigma)_{\sigma\in\Sigma}$ (or $\mathrm{S}(g'_\sigma)_{\sigma\in\Sigma}$) we assume that the operations g_σ (or g'_σ) are of arity $n+2r_\sigma$ (or $n+r_\sigma$) for some fixed $n \in \mathbb{N}_+$.

b) Usually the definition of primitive recursion also includes the case $n = 0$ in 1.3. Then however the operations g_σ are 0-ary constants for $\sigma \in \Sigma_0$ (cf. 1.3.1) which does not fit our convention not to consider 0-ary operations for clones (cf. 1.1). Nevertheless the restriction to $n \geq 1$ is no loss of generality: In fact, let $h : T \to T$ be the unary tree function obtained from (1.3.1) and (1.3.2) in case $n = 0$ with given constants $g_\sigma \in T$ for $r_\sigma = 0$ and operations $g_\sigma : T^{2r_\sigma} \to T$ for $r_\sigma \geq 1$. Further let $g'_\sigma : T^{1+2r_\sigma} \to T$ be the operations obtained from g_σ by adding a fictitious variable (at the first place), i.e. $g'_\sigma := g_\sigma(e_2^{1+2r_\sigma}, \ldots, e_{1+2r_\sigma}^{1+2r_\sigma})$ for $r_\sigma \geq 1$, and $g'_\sigma(a) := g_\sigma$ for $r_\sigma = 0$, $a \in T$. Then, for $h' := \mathrm{PR}(g'_\sigma)$ as given with 1.3, we have $h = h'(e, e)$ and $h' = h(e_2^2)$. Thus h belongs to a clone F if and only if h' does.

2 Clones Pol ϱ of tree functions closed under primitive recursion

The following theorem is the main result of this paper. It characterizes finitary relations ϱ over trees with the property that the clone $\mathrm{Pol}\,\varrho$ of tree functions is closed with respect to primitive recursion.

2.1 Theorem. *Let $m \in \mathbb{N}_+$ and let $\varrho \subseteq T^m$ be a reduced relation. Then the following conditions are equivalent:*

(i) $\langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{PR}} = \operatorname{Pol} \varrho$,

(ii) $\langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{S}} = \operatorname{Pol} \varrho$,

(iii) *there exist order ideals* $I_1, \dots, I_m \subseteq T$ *such that* $\varrho = I_1 \times \cdots \times I_m$.

2.2 Remarks. The tree functions in $\operatorname{Pol} \varrho$ with ϱ as in 2.1(iii) can easily be described:

$$\operatorname{Pol} \varrho = \bigcap_{j=1}^{m} \operatorname{Pol} I_j$$

where $\operatorname{Pol} I_j$ is the set of tree functions preserving the order ideal I_j (considered as unary relation on T), i.e. each $f \in \operatorname{Pol} I_j$ maps trees from I_j to trees which are again in I_j. By Theorem 2.1, every clone $\operatorname{Pol} I_j$ is PR-closed. Thus any intersection – finite or infinite – of clones of the form $\operatorname{Pol} I$ for some order ideal $I \subseteq T$ gives a PR-closed clone. Consequently the implication (iii) $\Longrightarrow$ (i) of Theorem 2.1 can be generalized to infinitary relations $\varrho = \prod_{j \in J} I_j$ (J being an infinite index set) because $\operatorname{Pol} \varrho = \bigcap_{j \in J} \operatorname{Pol} I_j$. However, the converse (i) $\Longrightarrow$ (iii) does not remain true for infinitary relations (cf. 3.3).

As mentioned in 1.1 the restriction to reduced relations is not a loss of generality, however it is crucial for the formulation of Theorem 2.1. If ϱ is not reduced then it is no longer a direct product of order ideals (note $I_1 \times I_1 \neq \{(x, x) \mid x \in I_1\}$).

In clone theory usually relations are even further reduced to relations without "fictitious components". In the context of Theorem 2.1 we have: the j-th component of $\varrho = I_1 \times \cdots \times I_m$ is fictitious iff the order ideal I_j is trivial, i.e., $I_j = T$. Relations which differ only in fictitious components determine the same clone $\operatorname{Pol} \varrho$.

In the remainder of this section we shall prove Theorem 2.1. Note that 2.1(i) $\Longrightarrow$ (ii) is trivial because $\operatorname{Pol} \varrho \subseteq \langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{S}} \subseteq \langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{PR}}$ (cf. 1.5.4). We start with the following more or less straightforward part:

Proof of 2.1 (iii) $\Longrightarrow$ (i).

Let $\varrho = I_1 \times \cdots \times I_m$ where $I_1, \dots, I_m \subseteq T$ are order ideals. Then $\operatorname{Pol} \varrho = \bigcap_{j=1}^{m} \operatorname{Pol} I_j$ (cf. 2.2) and (iii) $\Longrightarrow$ (i) of Theorem 2.1 follows from the following lemma.

2.3 Lemma. *Let* $I \subseteq T$ *be an order ideal. Then* $\langle\!\langle \operatorname{Pol} I \rangle\!\rangle_{\mathrm{PR}} = \operatorname{Pol} I$.

Proof. We have to show that $\operatorname{Pol} I$ is PR-closed. Let $n \in \mathbb{N}_+$ and, for every $\sigma \in \Sigma$, let g_σ be an $(n + 2r_\sigma)$-ary operation in $\operatorname{Pol} I$. We must show $h \in \operatorname{Pol} I$ for the $(n+1)$-ary operation $h := \operatorname{PR}(g_\sigma)_{\sigma \in \Sigma}$, i.e., $s_1, \dots, s_n, s \in I$ implies

$$h(s_1, \dots, s_n, s) \in I\,. \tag{2.3.1}$$

Thus let $s_1, \dots, s_n, s \in I$. We show 2.3.1 for $s \in T_k$ by induction on k:

For $s = \sigma \in \Sigma_0$ (i.e. $k = 0$) we have

$$h(s_1, \dots, s_n, s) = g_\sigma(s_1, \dots, s_n) \in I$$

because $g_\sigma \in \operatorname{Pol} I$ by assumption and $s_1, \ldots, s_n \in I$.

Now assume that 2.3.1 holds for every $s \in T_{k-1}$ $(k \geq 1)$. Let $s \in T_k$ be of the form $s = \sigma(t_1, \ldots, t_{r_\sigma})$, $r_\sigma \geq 1$. Then $t_1, \ldots, t_{r_\sigma} \in T_{k-1}$ and we get $h(s_1, \ldots, s_n, t_j) \in I$ for $j \in \{1, \ldots, r_\sigma\}$ by induction hypothesis. Note that $t_1, \ldots, t_{r_\sigma} \in I$ because I is an order ideal and $\sigma(t_1, \ldots, t_{r_\sigma}) = s \in I$ by assumption. Consequently

$$h(s_1, \ldots, s_n, s)$$
$$= g_\sigma(s_1, \ldots, s_n, t_1, \ldots, t_{r_\sigma}, h(s_1, \ldots, s_n, t_1), \ldots, h(s_1, \ldots, s_n, t_{r_\sigma})) \in I$$

because $g_\sigma \in \operatorname{Pol} I$ and every argument of g_σ belongs to I. □

The part (ii) $\Longrightarrow$ (iii) is crucial for the proof of Theorem 2.1. It is based on Proposition 2.5 which might be of independent interest (as well as Lemma 2.4): here we describe properties of the least S-closed clone $\langle\!\langle e \rangle\!\rangle_S$.

2.4 Lemma. *Let $s, t \in T$ and $s \leq t$. Then there exists an operation $f_{t,s} \in \langle\!\langle e \rangle\!\rangle_S$ such that $f_{t,s}(t) = s$.*

Proof. The case $t = s$ is trivial (take $f_{t,s} := e$). Thus assume $s < t$.

At first we consider the case $s \lessdot t$, i.e. t has the form $t = \delta(s_1, \ldots, s_k)$ with $s_i = s$ and $\delta \in \Sigma_k$ for some $k \geq 1$ and $i \in \{1, \ldots, k\}$. We construct a binary operation $h \in \langle\!\langle e \rangle\!\rangle_S$ with $h(t, t) = s$. Define $(g'_\sigma)_{\sigma \in \Sigma}$ as follows:

$$g'_\sigma := \begin{cases} e & \text{for } \sigma \in \Sigma_0 \\ e^{1+r}_{1+\min\{i,r\}} & \text{for } \sigma \in \Sigma_r,\ r \in \mathbb{N}_+ \end{cases}.$$

In particular we have $g'_\delta := e^{1+k}_{1+i}$. We note that for $\sigma \in \Sigma \setminus \{\delta\}$ the operations g'_σ will play no essential role in the following and could be chosen arbitrarily in $\langle\!\langle e \rangle\!\rangle_S$. Let $h := \mathrm{S}(g'_\sigma)_{\sigma \in \Sigma}$ (cf. 1.5.1 and 1.3 with $n = 1$). Since all g'_σ are projections (and therefore belong to $\langle e \rangle$) we have $h \in \langle\!\langle e \rangle\!\rangle_S$. Further we have

$$\begin{aligned} h(t, t) &= h(t, \delta(s_1, \ldots, s_i, \ldots, s_k)) \\ &= g'_\delta(t, s_1, \ldots, s_i, \ldots, s_k) \\ &= s_i = s\,. \end{aligned}$$

Choosing $f_{t,s} := h(e, e) \in \langle\!\langle e \rangle\!\rangle_S$ we have $f_{t,s}(t) = h(t, t) = s$ and consequently $f_{t,s} \in \langle\!\langle e \rangle\!\rangle_S$ exists for $s \lessdot t$.

Finally, if s is not a maximal subtree of t, then there exists a chain

$$s = s_l \lessdot \ldots \lessdot s_1 \lessdot s_0 = t$$

and as shown above for every $j \in \{0, 1, \ldots, l-1\}$, there exists $f_{s_j, s_{j+1}} \in \langle\!\langle e \rangle\!\rangle_S$ with $f_{s_j, s_{j+1}}(s_j) = s_{j+1}$; thus the composition

$$f := f_{s_{l-1}, s_l}(\ldots(f_{s_1, s_2}(f_{s_0, s_1}))\ldots)$$

satisfies $f(t) = s$ and belongs to $\langle\!\langle e \rangle\!\rangle_S$. □

2.5 Proposition. *For every $k \in \mathbb{N}$, $n, r \in \mathbb{N}_+$ and every family $(f_{b_1,\dots,b_r} \mid (b_1,\dots,b_r) \in T_k^r)$ of n-ary operations in $\langle\!\langle e \rangle\!\rangle_S$, there exists an $(n+r)$-ary operation $h \in \langle\!\langle e \rangle\!\rangle_S$ such that for all $a_1,\dots,a_n \in T$ and all $b_1,\dots,b_r \in T_k$ we have*

$$h(a_1,\dots,a_n,b_1,\dots,b_r) = f_{b_1,\dots,b_r}(a_1,\dots,a_n)\,. \tag{2.5.1}$$

Proof. In order to get h we construct (by double induction and using S-closure) auxiliary operations $h_{b_{i+1},\dots,b_r} \in \langle\!\langle e \rangle\!\rangle_S$ satisfying the following statement $R(k,n,r,i)$ for $i \in \{0,1,\dots,r\}$:

$$R(k,n,r,i) :\Longleftrightarrow \left[\begin{array}{l} \text{for every } b_{i+1},\dots,b_r \in T_k, \text{ there is an } (n+i)\text{-ary operation} \\ h_{b_{i+1},\dots,b_r} \in \langle\!\langle e \rangle\!\rangle_S \text{ such that for every } \tilde{a} \in T^n \text{ and } b_1,\dots,b_i \in \\ T_k\text{:} \\ \qquad h_{b_{i+1},\dots,b_r}(\tilde{a},b_1,\dots,b_i) = f_{b_1,\dots,b_r}(\tilde{a})\,, \qquad (2.5.2) \end{array}\right.$$

where

$$\begin{array}{l} (f_{b_1,\dots,b_r} \mid (b_1,\dots,b_r) \in T_k^r) \text{ is an} \\ \text{arbitrary family of } n\text{-ary operations in } \langle\!\langle e \rangle\!\rangle_S\,. \end{array} \tag{*}$$

We shall use the convention that $b_{i+1},\dots,b_r$ is the empty sequence for $i = r$ (i.e., $h_{b_{i+1},\dots,b_r}$ means h and $g_\sigma^{b_{i+1},\dots,b_r}$ below will mean g_σ in case $i = r$). Therefore, $R(k,n,r,r)$ is nothing else than the statement of Proposition 2.5, and we are done if we can prove

$$\forall k \in \mathbb{N}\, \forall n \in \mathbb{N}_+\, \forall r \in \mathbb{N}_+\, \forall (f_{b_1,\dots,b_r})_{\text{satisfying (*)}}\, \forall i \in \{0,1,\dots,r\} : R(k,n,r,i)\,. \tag{2.5.3}$$

We prove 2.5.3 by induction on k.

$\boxed{k=0}$ Let $n, r \in \mathbb{N}_+$ and $(f_{b_1,\dots,b_r} \mid (b_1,\dots,b_r) \in T_0^r)$ be a family of n-ary operations in $\langle\!\langle e \rangle\!\rangle_S$. Note that $T_0 = \Sigma_0$. We prove $R(0,n,r,i)$ for every $i \in \{0,1,\dots,r\}$ by induction on i.

$\boxed{i=0}$ $R(0,n,r,0)$ trivially holds by defining

$$h_{b_1,\dots,b_r} := f_{b_1,\dots,b_r} \in \langle\!\langle e \rangle\!\rangle_S\,. \tag{2.5.4}$$

$\boxed{i-1 \to i}$ Let $i \in \{1,\dots,r\}$ and assume that $R(0,n,r,i-1)$ holds, i.e., for every $b_i, b_{i+1},\dots,b_r \in T_0$ there exists $h_{b_i,b_{i+1},\dots,b_r} \in \langle\!\langle e \rangle\!\rangle_S$ fulfilling 2.5.2 for every $\tilde{a} \in T^n$ and $b_1,\dots,b_{i-1} \in T_0$.

Now, let $b_{i+1},\dots,b_r$ be arbitrary elements in T_0. For every $\sigma \in \Sigma$ we define the $(n+i-1+r_\sigma)$-ary operation $g_\sigma^{b_{i+1},\dots,b_r}$ as follows:

$$g_\sigma^{b_{i+1},\dots,b_r} := \begin{cases} h_{\sigma,b_{i+1},\dots,b_r} & \text{if } \sigma \in \Sigma_0 \\ e_1^{n+i-1+r_\sigma} & \text{otherwise} \end{cases}\,. \tag{2.5.5}$$

All these operations are in $\langle\!\langle e \rangle\!\rangle_S$ (for $\sigma \in \Sigma_0$ by induction hypothesis and otherwise by definition). Then the $(n+i)$-ary operation $h_{b_{i+1},\ldots,b_r}$ is defined by selection closure as follows:

$$h_{b_{i+1},\ldots,b_r} := \mathrm{S}(g_\sigma^{b_{i+1},\ldots,b_r})_{\sigma\in\Sigma}\,. \tag{2.5.6}$$

Thus $h_{b_{i+1},\ldots,b_r} \in \langle\!\langle e \rangle\!\rangle_S$, too. Moreover, condition 2.5.2 is satisfied because for every $b_1,\ldots,b_i \in T_0$ and $\tilde{a} \in T^n$ we have

$$\begin{aligned} h_{b_{i+1},\ldots,b_r}(\tilde{a},b_1,\ldots,b_i) &= g_{b_i}^{b_{i+1},\ldots,b_r}(\tilde{a},b_1,\ldots,b_{i-1}) && \text{by 2.5.6 (cf. 1.5.3)}\\ &= h_{b_i,b_{i+1},\ldots,b_r}(\tilde{a},b_1,\ldots,b_{i-1}) && \text{by 2.5.5}\\ &= f_{b_1,\ldots,b_r}(\tilde{a}) && \text{by } R(0,n,r,i-1). \end{aligned}$$

Thus $R(0,n,r,i)$ holds for all $n,r \in \mathbb{N}_+$ and $i \in \{0,1,\ldots,r\}$.

We can continue the induction on k.

$\boxed{k-1 \to k}$ Let $k \geq 1$. By induction we assume that $R(k-1,n',r',i')$ holds for every $n',r' \in \mathbb{N}_+$, every family $(f_{b_1,\ldots,b_{r'}} \mid (b_1,\ldots,b_{r'}) \in T_{k-1}^{r'})$ of n'-ary operations in $\langle\!\langle e \rangle\!\rangle_S$ and every $i' \in \{0,1,\ldots,r'\}$. Now, let $n,r \in \mathbb{N}_+$ and let $(f_{b_1,\ldots,b_r} \mid (b_1,\ldots,b_r) \in T_k^r)$ be a family of n-ary operations in $\langle\!\langle e \rangle\!\rangle_S$. We prove $R(k,n,r,i)$ for every $i \in \{0,1,\ldots,r\}$ by induction on i.

$\boxed{i=0}$ $R(k,n,r,0)$ trivially holds as in case $k=0$ and $i=0$ (cf. 2.5.4).

$\boxed{i-1 \to i}$ Let $i \in \{1,\ldots,r\}$ and assume that $R(k,n,r,i-1)$ holds. We prove $R(k,n,r,i)$. Thus let $b_{i+1},\ldots,b_r$ be arbitrary elements in T_k. Let $\sigma \in \Sigma$ and consider the family $(f^\sigma_{t_1,\ldots,t_{r_\sigma}} \mid t_1,\ldots,t_{r_\sigma} \in T_{k-1})$ of $(n+i-1)$-ary operations given by

$$f^\sigma_{t_1,\ldots,t_{r_\sigma}} := h_{\sigma(t_1,\ldots,t_{r_\sigma}),b_{i+1},\ldots,b_r}\,. \tag{2.5.7}$$

(Let us agree to include the case $r_\sigma = 0$ just by deleting all $t_1,\ldots,t_{r_\sigma}$ whenever they appear, i.e. $f^\sigma := h_{\sigma,b_{i+1},\ldots,b_r}$.) According to 2.5.7, all operations in this family exist and belong to $\langle\!\langle e \rangle\!\rangle_S$ by induction hypothesis $R(k,n,r,i-1)$ (note that $\sigma(t_1,\ldots,t_{r_\sigma}) \in T_k$ for $t_1,\ldots,t_{r_\sigma} \in T_{k-1}$). Now, by induction hypothesis $R(k-1,n+i-1,r_\sigma,r_\sigma)$, there exists an $(n+i-1+r_\sigma)$-ary operation $g_\sigma^{b_{i+1},\ldots,b_r} \in \langle\!\langle e \rangle\!\rangle_S$ such that

$$g_\sigma^{b_{i+1},\ldots,b_r}(a_1,\ldots,a_{n+i-1},t_1,\ldots,t_{r_\sigma}) = f^\sigma_{t_1,\ldots,t_{r_\sigma}}(a_1,\ldots,a_{n+i-1}) \tag{2.5.8}$$

for every $a_1,\ldots,a_{n+i-1} \in T$ and $t_1,\ldots,t_{r_\sigma} \in T_{k-1}$ (by our convention, it follows from 2.5.7 and 2.5.8 that $g_\sigma^{b_{i+1},\ldots,b_r}$ is defined for $r_\sigma = 0$ as in 2.5.5). Now, by S-closure, we define the $(n+i)$-ary operation $h_{b_{i+1},\ldots,b_r} = \mathrm{S}(g_\sigma^{b_{i+1},\ldots,b_r})_{\sigma\in\Sigma}$ as in 2.5.6. Thus $h_{b_{i+1},\ldots,b_r} \in \langle\!\langle e \rangle\!\rangle_S$. Moreover, let $b_1,\ldots,b_{i-1},b_i \in T_k$ and let b_i be of the form $b_i = \sigma(t_1,\ldots,t_{r_\sigma})$ for some $\sigma \in \Sigma$, then $t_1,\ldots,t_{r_\sigma} \in T_{k-1}$ and we have for every

$\tilde{a} \in T^n$

$$\begin{aligned} h_{b_{i+1},\dots,b_r}(\tilde{a}, b_1, \dots, b_i) &= g_\sigma^{b_{i+1},\dots,b_r}(\tilde{a}, b_1, \dots, b_{i-1}, t_1, \dots, t_{r_\sigma}) && \text{by 2.5.6} \\ &= f^\sigma_{t_1,\dots,t_{r_\sigma}}(\tilde{a}, b_1, \dots, b_{i-1}) && \text{by 2.5.8} \\ &= h_{\sigma(t_1,\dots,t_{r_\sigma}),b_{i+1},\dots,b_r}(\tilde{a}, b_1, \dots, b_{i-1}) && \text{by 2.5.7} \\ &= f_{b_1,\dots,b_r}(\tilde{a}) && \text{by } R(k,n,r,i-1). \end{aligned}$$

Thus $R(k,n,r,i)$ also holds. Both inductions (on i and on k) are done and 2.5.3 is proved. As mentioned before 2.5.3, this finishes the proof of Proposition 2.5. □

2.6 Remark. From the proofs it follows that 2.4 and 2.5 remain true if $\langle\!\langle e \rangle\!\rangle_S$ is substituted by $\langle\!\langle e \rangle\!\rangle_{\mathrm{PR}}$.

Now we can proceed with the

Proof of 2.1 (ii) $\Longrightarrow$ (iii).

Let ϱ be an m-ary reduced relation ($m \in \mathbb{N}_+$) and $\langle\!\langle \mathrm{Pol}\,\varrho \rangle\!\rangle_S = \mathrm{Pol}\,\varrho$. We have to show that ϱ is of the form as indicated in 2.1(iii). For $j \in \{1, \dots, m\}$ let

$$I_j := \{t_j \in T \mid \exists t_1, \dots, t_m : (t_1, \dots, t_j, \dots, t_m) \in \varrho\} \tag{2.6.1}$$

be the set of all j-th components of ϱ.

At first we show that every I_j is an order ideal. In fact, let $s < t$ and $t \in I_j$, then by Lemma 2.4 there exists an operation $f_{t,s} \in \langle\!\langle e \rangle\!\rangle_S \subseteq \langle\!\langle \mathrm{Pol}\,\varrho \rangle\!\rangle_S$ with $f_{t,s}(t) = s$. Applying $f_{t,s}$ to an m-tuple in ϱ with t in its j-th component we get an m-tuple in ϱ with s in its j-th component, i.e. $s \in I_j$. Thus I_j is an order ideal.

Now we are going to show that $\varrho = I_1 \times \cdots \times I_m$ which will finish the proof of 2.1(ii) $\Longrightarrow$ (iii) (and thus finish the proof of Theorem 2.1).

Let $t_{11} \in I_1, \dots, t_{mm} \in I_m$. We have to show $(t_{11}, \dots, t_{mm}) \in \varrho$. By 2.6.1 there exist elements $(t_{1j}, \dots, t_{jj}, \dots, t_{mj}) \in \varrho$ with t_{jj} as j-th component ($j \in \{1, \dots, m\}$), which we represent as columns of an $(m \times m)$-matrix. If not all rows are different then we can add some further, say $r - m$, columns, which we denote by $(t_{1j}, \dots, t_{mj}) \in \varrho$, $j = m+1, \dots, r$, such that now all rows of the corresponding matrix A are pairwise different (this is possible because ϱ is a reduced relation):

$$A = \begin{pmatrix} t_{11} & \dots & t_{1m} & \dots & t_{1r} \\ \vdots & \ddots & \vdots & \dots & \vdots \\ t_{m1} & \dots & t_{mm} & \dots & t_{mr} \end{pmatrix}.$$

Let k be the maximal height of all trees t_{ij} ($i = 1, \dots, m$, $j = 1, \dots, r$). By Proposition 2.5 there exists a $2r$-ary operation $h \in \langle\!\langle e \rangle\!\rangle_S$ such that

$$h(a_1, \dots, a_r, t_{i1}, \dots, t_{ir}) = a_i \tag{2.6.2}$$

for every $i \in \{1,\dots,m\}$ and $a_1,\dots,a_r \in T$. In fact, in 2.5 take $n = r$, and let $f_{b_1,\dots,b_r} = e_i^r$ for $(b_1,\dots,b_r) = (t_{i1},\dots,t_{ir})$, $i \in \{1,\dots,m\}$, and arbitrary in $\langle\!\langle e \rangle\!\rangle_S$ otherwise (e.g. $f_{b_1,\dots,b_r} := e_1^r$). Consequently, the operation f defined by

$$f(a_1,\dots,a_r) := h(a_1,\dots,a_r,a_1,\dots,a_r) \tag{2.6.3}$$

(i.e. $f = h(e_1^r,\dots,e_r^r,e_1^r,\dots,e_r^r)$) also belongs to $\langle\!\langle e \rangle\!\rangle_S \subseteq \langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_S = \operatorname{Pol} \varrho$.

Applying f to the matrix A row-wise we obtain (cf. 2.6.2 and 2.6.3) the m-tuple $(t_{11},\dots,t_{mm})$; it must belong to ϱ because all columns of A are in ϱ and f preserves ϱ. □

3 Further research and remarks

3.1. Connections between closure operators. For $F \subseteq \operatorname{Op}(T)$, let $\operatorname{PR}(F)$ denote the set which contains F and all tree functions $h = \operatorname{PR}(g_\sigma)_{\sigma\in\Sigma}$ with $g_\sigma \in F$. Further let $\operatorname{PR}^n(F) := \bigcup_{i=1}^n \operatorname{PR}^i(F)$, where $\operatorname{PR}^1(F) := \operatorname{PR}(F)$ and $\operatorname{PR}^{i+1}(F) := \operatorname{PR}(\operatorname{PR}^i(F))$ $(i \in \mathbb{N}_+)$. Then

$$\operatorname{PR}^* F := \bigcup_{i=1}^{\infty} \operatorname{PR}^i(F)$$

is the least PR-closed set of tree functions containing F. The mapping $F \mapsto \operatorname{PR}^* F$ is a closure operator as well as $F \mapsto \langle F \rangle$ and $F \mapsto \langle\!\langle F \rangle\!\rangle_{\mathrm{PR}}$. By definition (cf. 1.3) we have

$$\langle\langle F\rangle\!\rangle_{\mathrm{PR}}\rangle = \langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} \qquad \langle\!\langle\langle F\rangle\rangle\!\rangle_{\mathrm{PR}} = \langle\!\langle F\rangle\!\rangle_{\mathrm{PR}}$$
$$\operatorname{PR}^*\langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} = \langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} \qquad \langle\!\langle \operatorname{PR}^* F\rangle\!\rangle_{\mathrm{PR}} = \langle\!\langle F\rangle\!\rangle_{\mathrm{PR}}.$$

It is still an open question how the iterations of the operators PR^* or PR and $\langle . \rangle$ behave in general. E.g. do we have

$$\langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} = \operatorname{PR}^*\langle \dots \langle \operatorname{PR}^*\langle \operatorname{PR}^*\langle \underbrace{F\rangle\rangle\rangle \dots \rangle}_{n} \tag{3.1.1}$$

$$\text{or} \quad \langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} = \operatorname{PR}\langle \dots \langle \operatorname{PR}\langle \operatorname{PR}\langle \underbrace{F\rangle\rangle\rangle \dots \rangle}_{n} \tag{3.1.2}$$

for a fixed finite number n of iterations of PR^* or PR and $\langle . \rangle$? Or, if the answer is negative, for which F does this iteration stabilize after a finite number of steps?

For instance, if $F = \operatorname{PREC}_\Sigma$ (cf. 1.4b), then obviously $\langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} = F$. On the other hand, if $F = F_{\mathrm{base}}$ (cf. 1.4b), then $\langle\!\langle F\rangle\!\rangle_{\mathrm{PR}} = \bigcup_{n=1}^{\infty}\langle \operatorname{PR}\rangle^n(F) = \bigcup_{n=1}^{\infty}\langle \operatorname{PR}^*\rangle^n(F)$ where $\langle \operatorname{PR}\rangle^n(F)$ and $\langle \operatorname{PR}^*\rangle^n(F)$ is an abbreviation of the right side of 3.1.2 and 3.1.1, respectively. Here, in general, we really need the union over all $n \in \mathbb{N}_+$ (the first union reflects the GRZEGORCZYK-hierarchy).

Analogous questions arise with the closure $\langle\!\langle F\rangle\!\rangle_S$ instead of $\langle\!\langle F\rangle\!\rangle_{\mathrm{PR}}$.

3.2. Specialization to $\mathbb{N}$. Let us specialize the signature Σ to the signature of the natural numbers $\langle \mathbb{N}; \mathrm{succ}, 0\rangle$ with successor function $n \mapsto \mathrm{succ}\, n$ ($\mathrm{succ}\, n = n + 1$) and the constant 0. Then $\Sigma = \{\mathrm{succ}, 0\}$ and the set T of trees over σ (cf. 1.2) can be identified with $\mathbb{N}$. Primitive recursion (as defined in 1.3) is just the usual primitive recursion for operations on natural numbers. This case was studied in detail in, e.g., [Pét57]. In [Sem02] Theorem 2.1 has been proved for this particular signature. Then an order ideal in $\mathbb{N}$ is either a principal ideal or the whole set $\mathbb{N}$. Thus (cf. Remark 2.2), in order to describe PR-closed clones of the form $\mathrm{Pol}\,\varrho$, one can restrict to relations ϱ that are the product of principal ideals $I_j = \{0, 1, \dots, a_j\}$ of $\mathbb{N}$. Moreover, in [Sem02], it was proved that if Q is a set of finitary reduced relations such that $\langle\!\langle \mathrm{Pol}\, Q\rangle\!\rangle_{\mathrm{PR}} = \mathrm{Pol}\, Q$, then $\langle\!\langle \mathrm{Pol}\,\varrho\rangle\!\rangle_{\mathrm{PR}} = \mathrm{Pol}\,\varrho$ for each relation $\varrho \in Q$. Further, the partially ordered set of clones of the form $\mathrm{Pol}\, Q$, where Q is as above, is isomorphic to the lattice of all subsets of $\mathbb{N}$ ordered by inclusion. In addition, it was shown what happens if one considers only recursive or primitive recursive polymorphisms instead of all possible polymorphisms.

It is still unknown how these results can be generalized to arbitrary tree functions.

3.3. Generalization to infinitary relations. Theorem 2.1 shows that PR-closed classes of tree functions characterizable by a finitary relation have a very simple structure (cf. 2.2).

However, it was also mentioned in 2.2 that for infinitary relations ϱ, $\langle\!\langle \mathrm{Pol}\,\varrho\rangle\!\rangle_{\mathrm{PR}} = \mathrm{Pol}\,\varrho$ in general does not imply that ϱ is the direct product of order ideals. To give an example, consider the signature $\Sigma = \{\mathrm{succ}, 0\}$ as in 3.2 and the infinitary ($|\mathbb{N}|$-ary) relation $\varrho \subseteq \mathbb{N}^{\mathbb{N}}$ defined by

$$\varrho := \{g \in \mathbb{N}^{\mathbb{N}} \mid g \text{ is a unary primitive recursive function}\}.$$

Claim: $\mathrm{Pol}\,\varrho$ *is the set of all primitive recursive functions over* $\mathbb{N}$.

Proof of the claim:

At first recall that an operation $f : \mathbb{N}^n \to \mathbb{N}$ preserves ϱ iff $g_1, \dots, g_n \in \varrho$ implies $f(g_1, \dots, g_n) \in \varrho$ where $f(g_1, \dots, g_n) : \mathbb{N} \to \mathbb{N}$ is the composition defined by

$$f(g_1, \dots, g_n)(x) := f(g_1(x), \dots, g_n(x)) \tag{3.3.1}$$

for $x \in \mathbb{N}$; this is the obvious generalization of 1.1.3 to the infinitary case.

By definition, the composition $f(g_1, \dots, g_n)$ of primitive recursive functions $f, g_1, \dots, g_n$ is again primitive recursive. Thus every primitive recursive $f : \mathbb{N}^n \to \mathbb{N}$ preserves ϱ, i.e., $\mathrm{Pol}\,\varrho$ contains the set of all primitive recursive functions on $\mathbb{N}$.

Conversely, every $f \in \mathrm{Pol}\,\varrho$ is primitive recursive.

Indeed, let $f \in \mathrm{Pol}\,\varrho$ be unary, then $f(e) \in \varrho$ because $e \in \varrho$; thus $f = f(e)$ is also primitive recursive.

Now suppose that $f \in \mathrm{Pol}\,\varrho$ is n-ary, where $n > 1$. In this case, we use the fact (see, for example, [Pét57]) that there exist unary primitive recursive functions g_1, g_2 and a binary primitive recursive function h such that $g_1(h(x_1, x_2)) = x_1$ and

$g_2(h(x_1, x_2)) = x_2$ for all $x_1, x_2 \in \mathbb{N}$. Now define the functions

$$\begin{aligned} c_2(x_1, x_2) &:= h(x_1, x_2), & c_{i+1}(x_1, \ldots, x_{i+1}) &:= h(c_i(x_1, \ldots, x_i), x_{i+1}) && \text{for } i > 1, \\ l_1(x) &:= g_1(x), & l_{j+1}(x) &:= g_1(l_j(x)) && \text{for } j \in \mathbb{N}_+, \\ r_1(x) &:= g_2(x), & r_{j+1}(x) &:= g_2(l_j(x)) && \text{for } j \in \mathbb{N}_+. \end{aligned}$$

Clearly, the functions c_{j+1}, l_j, and r_j are primitive recursive for $j \in \mathbb{N}_+$. It can easily be checked that

$$r_j(c_i(x_1, \ldots, x_i)) = x_{i-j+1} \qquad \text{for } j \in \mathbb{N}_+ \text{ and } i > j, \tag{3.3.2}$$

$$l_j(c_{j+1}(x_1, \ldots, x_{j+1})) = x_1 \qquad \text{for } j \in \mathbb{N}_+. \tag{3.3.3}$$

Since $f \in \operatorname{Pol} \varrho$ and $l_{n-1}, r_{n-1}, r_{n-2}, \ldots, r_1 \in \varrho$ we have

$$g := f(l_{n-1}, r_{n-1}, r_{n-2}, \ldots, r_1) \in \varrho,$$

i.e., g is primitive recursive. Using 3.3.2 and 3.3.3, we get $f(x_1, \ldots, x_n) = g(c_n(x_1, \ldots, x_n))$. Hence f is primitive recursive because it is a composition of the primitive recursive functions g and c_n. □

From the above claim we know $\langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{PR}} = \operatorname{Pol} \varrho$. On the other hand, ϱ is not the direct product of order ideals of $\mathbb{N}$.

Thus ϱ is a counterexample to the straightforward generalization of Theorem 2.1 to infinitary relations and there arises the problem how this theorem could be extended appropriately to infinitary relations.

3.4. C-closed clones. A clone F of tree functions shall be called C-*closed* if it is PR-closed (i.e. $\langle\!\langle F \rangle\!\rangle_{\mathrm{PR}} = F$) and closed under *iteration*. The latter means (cf. e.g. [Hup78]) that if $f, h, g_1, \ldots, g_n$ are n-ary tree functions in F then every tree function $k : T^n \to T$, which is definable by a program of the following form for some $t \in T$, must also belong to F.

$$\begin{aligned} &\text{WHILE} && f(x_1, \ldots, x_n) \neq t \\ &\text{DO} && x_1 := g_1(x_1, \ldots, x_n); \\ & && \vdots \\ & && x_n := g_n(x_1, \ldots, x_n) \\ &\text{OD;} \\ &\text{OUTPUT} && h(x_1, \ldots, x_n) \end{aligned}$$

Note that here we consider only total operations k defined by iteration (while in [Hup78] also partial operations are allowed). The C-closure is stronger than the PR-closure, e.g., let $\langle\!\langle F \rangle\!\rangle_{\mathrm{C}}$ denote the smallest C-closed clone containing F, then, as shown in [Hup78], $\langle\!\langle F_{\mathrm{base}} \rangle\!\rangle_{\mathrm{C}}$ is the set of all computable tree functions (for F_{base} see 1.4b).

Nevertheless, as it was pointed out by the referee, clones of the form $\operatorname{Pol} \varrho$ with ϱ as in Theorem 2.1(iii) are C-closed. Thus Theorem 2.1 can be extended by an additional equivalent condition

(iv) $\langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{C}} = \operatorname{Pol} \varrho$.

Obviously $\langle\!\langle F \rangle\!\rangle_{\mathrm{S}} \subseteq \langle\!\langle F \rangle\!\rangle_{\mathrm{PR}} \subseteq \langle\!\langle F \rangle\!\rangle_{\mathrm{C}}$. It is an open question to characterize the least and (if it exists) the largest closure which agrees with $\langle\!\langle F \rangle\!\rangle_{\mathrm{PR}}$ (or equivalently with $\langle\!\langle F \rangle\!\rangle_{\mathrm{S}}$, $\langle\!\langle F \rangle\!\rangle_{\mathrm{C}}$) for clones of the form $\operatorname{Pol} \varrho$. More precisely, let $\mathcal{K}$ be the class of all closure operators $K : \mathfrak{P}(\operatorname{Op}(T)) \to \mathfrak{P}(\operatorname{Op}(T))$ on tree functions such that $K(\operatorname{Pol} \varrho) = \langle\!\langle \operatorname{Pol} \varrho \rangle\!\rangle_{\mathrm{PR}}$ for all finitary relations $\varrho \subseteq T^m$ ($m \in \mathbb{N}_+$). Then

$$K_0 : F \mapsto \bigcap_{K \in \mathcal{K}} K(F)$$

is the least closure operator in $\mathcal{K}$, but it is not clear how it could be characterized internally. Moreover, does there exist a largest closure operator in $\mathcal{K}$?

3.5. Further generalizations. The preservation (or invariance) property (cf. 1.1) constitutes a Galois connection between sets of operations and relations. There are many generalizations and modifications of this Galois connection changing the operations and/or relations under consideration (see e.g. [Pös01]). A systematic investigation of operations, their invariant relations and various closures, which are of special interest for computer science, would be desirable. In connection with tree functions and primitive recursion the class of partial tree functions may be of particular interest. Then the C-closure (cf. 3.4) might play the role of the PR-closure in Theorem 2.1. However note that the C-closure still can be extended further: the condition that the operation f used in the iteration program in 3.4 belongs to F can be dropped (without changing the property that k preserves ϱ whenever $h, g_1, \ldots, g_n$ do).

Acknowledgements. The authors thank the referee for valuable hints and remarks, e.g. for drawing our attention to the S-closure and C-closure (discussed in 1.5 and 3.4).

References

[EngV91] J. Engelfriet and H. Vogler. *Modular tree transducers.* Theoretical Computer Science 78 (1991), 267–303.

[FülHVV93] Z. Fülöp, F. Herrmann, S. Vágvölgyi, and H. Vogler. *Tree transducers with external functions.* Theoret. Comput. Sci. 108 (1993), 185–236.

[Hup78] U.L. Hupbach. *Rekursive Funktionen in mehrsortigen Peano-Algebren.* EIK, 14 (1978), 491–506.

[Kla84] H.A. Klaeren. *A constructive method for abstract algebraic software specification.* Theoret. Comput. Sci. 30 (1984), 139–204.

[Pét57] R. Péter. *Rekursive Funktionen.* Akademie-Verlag 1957.

[Pös80] R. Pöschel. *A general Galois theory for operations and relations and concrete characterization of related algebraic structures.* Report R-01/80, Zentralinstitut für Mathematik und Mechanik, Akademie der Wissenschaften der DDR, Berlin, 1980. (with German and Russian summaries), 101 pp.

[Pös01] R. Pöschel. *Galois connections for operations and relations.* Technical Report MATH-AL-8-2001, TU Dresden, October 2001. (29 pages).

[PösK79] R. Pöschel and L.A. Kalužnin. *Funktionen- und Relationenalgebren.* Deutscher Verlag der Wissenschaften, Berlin, 1979. Birkhäuser Verlag Basel, Math. Reihe Bd. 67, 1979.

[Sem02] A.P. Semigrodskikh. *On properties preserved by primitive recursion.* Technical Report MATH-AL-12-2002, TU Dresden, August 2002. (22 pages).

Received December, 2002

Acta Cybernetica **16** (2004) 427–441.

Notes on the properties of dynamic programming used in direct load control

Isto Aho*

Abstract

We analyze a dynamic programming (DP) solution for cutting overload in electricity consumption. We are able to considerably improve the earlier DP algorithms. Our improvements make the method practical so that it can be used more often or, alternatively, new state variables can be added into the state space to make the results more accurate.

1 Introduction

The shortage of electricity may cause a supplier to use direct load control (DLC); the supplier may turn off the electricity from some of its customers or may start generators to meet the demand. The goal is to minimize the losses by buying the minimum amount of electricity from other suppliers to cover the demand after DLC. A typical example of a control group is a residential appliance with electricity heaters or air conditioners. When the supplier controls the devices, a consumption peak called payback appears after the control period when the devices go back to their normal state [1, 17].

Different solution methods for DLC include, e.g., DP [1, 7, 8, 18], linear programming (LP) [12, 14], heuristics [1, 3], enumerative methods [8], and hybrids of LP and DP [13]. Objectives include load reduction minimization [7, 8, 18], peak load minimization [12, 13], production cost minimization [8, 18], and profit maximization [14]. Some of the decisions can be left to the customers [9]. DLC is often combined with unit commitment and economic dispatch, and the applied methods include, e.g., DP [4, 5, 6, 11] and evolutionary strategies [10]. See also [19].

Our method is related to that of [1, 7, 8, 14]. The present work improves the results of [1] by focusing on DP. In our model, (small) electricity suppliers group their customers based on the payback behavior: the payback shapes and other properties to be presented are for groups. Thus, the suppliers end up with a small number of controllable and prioritized groups. Our objective is between load reduction minimization and peak load minimization, and it differs from the objectives presented in the literature. We assume that the suppliers mainly buy

*Dept. of Computer and Information Sciences, University of Tampere, Tampere, Finland, Isto.Aho@uta.fi

the electricity they distribute. The above assumptions are realistic in Finland and in some other European countries with small suppliers.

Purchase transactions of electricity and own production give optimum level to be resold at each hour, while load over the optimum level has high price. If demand is higher than our predefined level, we want to cut (expensive) "over loads". We also use purchasing and reselling prices (time-of-use rates). Our solution can use different objectives and, e.g., energy storages of [15, 16], without major modifications.

Sections 2 and 3 describe the model, optimization problem and the DP solution. The main results concern the "wait states" and "alternative states" (state variables) needed in DP. We build up a hierarchy of DP solutions so that it is possible to choose between fast and inaccurate and slow but more accurate methods. Section 4 shows how the number of wait states can be decreased to be about half of the number used in [1]. State space can also be decreased with the multi-pass DP of [18], but then one should be able to relax some constraints. Section 4 includes also the use a fourth state variable (alternative states). Tests are reported in Section 5.

2 Control, payback, restrictions and goal function

The model given below is a slight simplification of the model used in [1]. Table 1 contains relevant symbols used in this work.

An *interval* $[a, b]$ is the set $a, a+1, \ldots, b$ $(a < b)$ of integers. The *length* of an interval $[a, b]$ is $b - a + 1$. A *clipping situation* $\mathbf{s}$ is a vector $s_0, s_1 \ldots, s_N$ $(N > 0)$ of reals representing the difference between electricity demand, and electricity production and purchases in time interval $[0, N]$. The domain $[0, N]$ is called the *optimization interval* and values s_i are called either *overload* or *underload*. Overload represents a situation where the demand is higher than combined production and

Table 1: Used symbols.

Clipping situation	$\mathbf{s} = [s_0, s_1, \ldots, s_N]$	Set of controls	$\mathbf{C}$
Prices	$\mathbf{p} = [p_0, p_1, \ldots, p_N]$	Control capacity	C^c
Revenue	$\mathbf{r} = [r_0, r_1, \ldots, r_N]$	Control length	C^l
Length of hour	h^l	Resting time	C^r
Time interval or control	$[a, b]$	Minimum control length	C^m
Loss of incomes	$R(\mathbf{s})$	Maximum control length	C^M
Optimal control plan	$R^*(\mathbf{s})$	Maximum control times	C^T
Dynamic forward recursion	$R'(\mathbf{s}, S', k+1)$	Control time	C^t
Stage change	$R''(\mathbf{s}, S', S, k+1)$	Length of payback	P^l
Wait state	W	Amount of payback[a]	P^c
Alternative state	A	Impact of a control	$I([a, b], \mathbf{s})(k)$
State (3 variable)	$S = (C^t, W, C^l)$	Impacts of all controls	$I'(\mathbf{C}, \mathbf{s})$
State (4 variable)	$S = (C^t, A, W, C^l)$		

[a] Amount of payback corresponds to capacity explaining the c-superscript.

electricity purchases ($s_i \geq 0$), and underload represents a situation where combined production and purchases of electricity is above the level of consumption ($s_i \leq 0$). The element i of optimization interval $[0, N]$ is called a *time point*. The phrase "time point i" is also used for the interval $[i, i]$.

A time point i with overload has a positive real p_i called the *price* (buying price of electricity). Similarly, a time point i with underload has a positive real r_i, the *revenue* (selling price of electricity). The *overload interval* is an interval $[a, b] \subseteq [0, N]$ with overload at every time point $i \in [a, b]$. The clipping situation is partitioned into *hours* $0 = a_1, a_2, \ldots, a_{n+1} = N$, of equal length, i.e., $a_{i+1} - a_i = a_i - a_{i-1}$ $(2 \leq i \leq n)$. The length $a_{i+1} - a_i$ of an hour is denoted by h^l. Hour i refers to the interval $[a_i, a_{i+1} - 1]$. Overloads (underloads), revenues and prices do not change during an hour, because of the electricity trading system. Thus, we have $s_j = s_{j+1}$, $p_j = p_{j+1}$, and $r_j = r_{j+1}$, where $j \in [a_{i-1}, a_i - 1]$ $(2 \leq i \leq n)$.

The total loss is

$$R(\mathrm{s}) = \sum_{i \in [0,N]} K(i, s_i), \quad \text{where} \quad K(i, s_i) = \begin{cases} -p_i s_i, & \text{if } s_i \geq 0, \\ r_i s_i, & \text{otherwise.} \end{cases} \tag{1}$$

Note that $K(i, s_i) \leq 0$. If there is underload, we lose income (revenue) and if there is overload, we have to pay some extra. We count the money lost, so its best possible value is 0.

A *group* is used to decrease the overload with a *control* made for interval $[a, b]$ by turning electricity off (an auxiliary generator corresponds to a group). The *controlling capacity* of a group, denoted by C^c, is the amount the group can decrease the load in an hour. The hours $[a_i, a, a_{i+1}, \ldots, a_{j-1}, b, a_j]$, where $a_i \leq a < a_{i+1}$ and $a_{j-1} < b \leq a_j$ (and $a < b$), are affected by a control. *Control amount* is the product of the controlling capacity C^c and of the control length $b - a + 1$.

Function $P^l : \mathbb{N} \to \mathbb{N}$ maps the control length $b - a + 1$ to the length of a payback and function $P^c : 2^{\mathbb{N}} \times \mathbb{N} \to \mathbb{R}$ describes the amount of the payback of control $[a, b]$ at time i. We always have $P^c([a, b], i) \geq 0$, where $i \in [b + 1, b + P^l(b - a + 1)]$, and otherwise $P^c([a, b], i) = 0$. Further, "in practice" we have

$$\sum_{k \in [b+1, b+P^l(b-a+1)]} P^c([a, b], k) \leq C^c(b - a + 1)$$

meaning that a payback does not exceed the control amount.

Next we show the impact of a control $[a, b]$ and its payback to clipping situation s as a function I (functions I_1 and I_2 used in I are defined below). The hours to be affected are $[a_i, a, b, a_j, b + P^l(b - a + 1), a_l]$. By function

$$I\big([a, b], \mathrm{s}\big)(k) = \begin{cases} s_k, & \text{if } 0 \leq k < a_i, \text{ or } a_l \leq k \leq m, \\ s_k + I_1\big([a, b]\big)(k), & \text{if } a_i \leq k < a_{j-1}, \\ s_k + \big(I_1 + I_2\big)\big([a, b]\big)(k), & \text{if } a_{j-1} \leq k < a_j, \\ s_k + I_2\big([a, b]\big)(k), & \text{if } a_j \leq k < a_l \end{cases} \tag{2}$$

$(0 \leq k \leq m)$ we obtain the control amount of a control $[a,b]$ into the clipping situation. The first line leaves the unaffected hours as they are and the second line calculates the effects of control. The third line is for both the control and payback calculation, and the fourth line calculates the effects of payback. By $I([a,b],\mathrm{s})$ we mean the new clipping situation obtained after control $[a,b]$.

The effects of control part having no payback are calculated with I_1:

$$I_1([a,b])(k) = \begin{cases} -C^c(a_{i+1}-1-a+1)/h^l, & \text{if } a_i \leq k < a_{i+1}, \\ -C^c, & \text{if } a_{i+1} \leq k < a_{j-1}, \\ -C^c(b-a_{j-1})/h^l, & \text{if } a_{j-1} \leq k < a_j. \end{cases} \tag{3}$$

The second line is for the hours between the starting and stopping hours, if any. The first and the third lines handle the hours where the control starts and stops. These hours may have partial control (in contrast to a full control lasting the whole hour). Controls decrease the overload. Paybacks are calculated with I_2:

$$I_2([a,b])(k) = \begin{cases} \displaystyle\sum_{k'=b+1}^{a_j-1} \frac{P^c([a,b],k')}{h^l}, & \text{if } a_{j-1} \leq k < a_j, \\ \displaystyle\sum_{k''=a_{k'}}^{a_{k'+1}-1} \frac{P^c([a,b],k'')}{h^l}, & \text{if } j \leq k' \leq l-1 \\ & \text{and } a_{k'} \leq k < a_{k'+1}, \\ \displaystyle\sum_{k'=a_{l-1}}^{a_l-1} \frac{P^c([a,b],k')}{h^l}, & \text{if } a_{l-1} \leq k \\ & < b + P^l(b-a+1). \end{cases} \tag{4}$$

The payback starts in the first line, and in the third line we calculate the last moments having payback. (Both may involve partial hours.) The second line calculates the payback for hours, where each time point will get some payback. The payback increases the overload.

It would simplify formulas (2)–(4) a bit if we were not to hourly even out the effects. Another alternative is to let the overloads and underloads vary within the hours and even out the loads when calculating the results. If the control starts and stops in the same hour, we cannot directly apply (2). In this situation we calculate the effects for the first hour with

$$s_k - C^c(b-a+1)/h^l + \sum_{k'=b+1}^{a_j-1} P^c([a,b],k')/h^l, \text{ where } a_{j-1} \leq k < a_j, \tag{5}$$

and the rest of the payback is calculated with the second line of I_2. If the payback starts and stops in the same hour, we have to make a correction similar to (5). Energy storage capability is similar to payback: energy storing appears before control while payback appears after the control.

Figure 1 shows two examples of a control. The vertical lines indicate hours. The dotted line is a clipping situation without control and the straight line is a clipping situation with control. The left picture shows the advantage of a control: payback can "move" the overload to the next hour where the overload is cheaper.

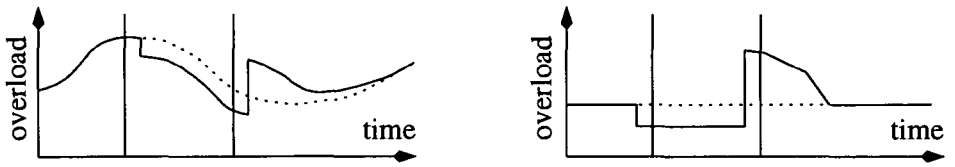

Figure 1: Realistic and theoretic clipping situation with a control.

The combined effect of all controls $\mathbf{C}$ can be calculated recursively by the function

$$I'(\mathbf{C},\mathbf{s}) = \begin{cases} I'(\mathbf{C}-[a,b], I([a,b],\mathbf{s})), & \text{if } [a,b]\in\mathbf{C}\ (\neq\emptyset), \\ \mathbf{s}, & \text{if } \mathbf{C}=\emptyset. \end{cases} \tag{6}$$

When all the effects of controls have been calculated, we can use (1) to find out the value of the new clipping situation.

Next we look at the restrictions. First, the controls must be separate such that for all $[a,b],[c,d]\in\mathbf{C}$ we have

$$[a,b]\neq[c,d] \Rightarrow [a,b]\cap[c,d]=\emptyset. \tag{7}$$

Further, during *resting time* it is not allowed to start a new control. Function $C^r : \mathbb{N}\to\mathbb{N}$ is increasing and it maps the length of a control to the length of a resting time. So, for all $[a,b],[c,d]\in\mathbf{C}$ we have

$$[a,b]\cap[c,d]=\emptyset \Rightarrow [b+1, b+C^r(b-a+1)]\cap[c,d]=\emptyset. \tag{8}$$

Note that a new control can be started even if the payback still occurs, provided that the resting time does not overlap with the new control. Usually, the resting time is used to prevent a new control to start in the beginning of the payback, when the need for extra electricity is the largest. If we start a new control at the end of a payback, the change in the payback of the new control is so small that it is not usually taken into account.

We also need the *minimum* and *maximum* control times C^m and C^M, respectively, which bound the length of control as

$$C^m \leq b-a+1 \leq C^M. \tag{9}$$

Sometimes we also restrict the number of control times $\sum_{[a,b]\in\mathbf{C}} 1$ by C^T, a positive integer.

We can suppose that at every time point k the price factor p_k is (much) larger than the revenue r_k. By making controls we can affect the clipping situation, so the optimization problem can be given in the form

$$\max_{\mathbf{C}} \sum_{k=0}^{N} R(k, I(\mathbf{C},\mathbf{s})) \tag{10}$$

with restrictions (7)–(9).

3 A DP solution

The problem (10) can be solved with DP by using two state variables corresponding to the control times and control length [1, 8, 7, 19]. The tests in [1] indicated that it is possible to add at least one state variable to improve the results. In this work we use the state variables *control time* C^t, *wait state* W and *control length* C^l. As a result, we have a slower but more accurate system than those with two state variables. The state variables are defined in finite integer domain.

In the state space we need the control length C^l, so that DP can form the optimal control length and at the same time fulfill the restriction (9). In addition to control length, C^l also contains the resting time. Without the control times, DP complying with conditions (7)–(9) would find only one control. With these three state variables we have one state of stage $k \in [0, N]$ as a triple $(C^t, W, C^l)(k)$. The phrase "stage k" refers to a time point. A system in state (C^t, W, C^l) is defined to be the C^tth control of length C^l with W time points delay before its start. Our tests demonstrate that the three variable solution does not give the optimal solution, if the the group has payback (see Section 5).

In practice we have to determine upper bound for the number of wait states W. Theorem 2 gives an upper bound for W when the group does not have payback. If the group does have payback, we assume that W can have $h^l - 1$ (h^l is the length of an hour) different values. We also test other possibilities, see Section 5.

We denote $S = (C^t, W, C^l)$ and $S' = (C'^t, W', C'^l)$. The variables with primes are "new" ones and the plain variables are "old", when we form the connections between the "new" stage $k+1$ and the "old" stage k. Function

$$R''(\mathrm{s}, S', S, k+1) = \begin{cases} 0, & \text{when (14)–(17),} \\ -P', & \text{when (18),} \\ R(I([k - C^t, k], \mathrm{s})) - R(\mathrm{s}), & \text{when (19),} \\ -\infty, & \text{otherwise,} \end{cases} \tag{11}$$

gives the change in the value, when moving from state (C^t, W, C^l) of stage k into state (C'^t, W', C'^l) of stage $k+1$. The first line is used when the value does not change. The second line is used, when we start a new control and the third line is applied, when we make a decision about the best control. The cost of making a control is denoted by P'. The last line is used with every other values of the variables S and S'. They are impossible since they do not have any reasonable real world interpretation.

The dynamic forward recursion equation is

$$R'(\mathrm{s}, S', k+1) = \max_S \left(R'(\mathrm{s}, S, k) + R''(\mathrm{s}, S', S, k+1) \right) \tag{12}$$

and

$$R'(\mathrm{s}, (C^t, W, C^l), 0) = \begin{cases} R(\mathrm{s}), & \text{when } C^t = W = C^l = 0, \\ -\infty, & \text{otherwise.} \end{cases} \tag{13}$$

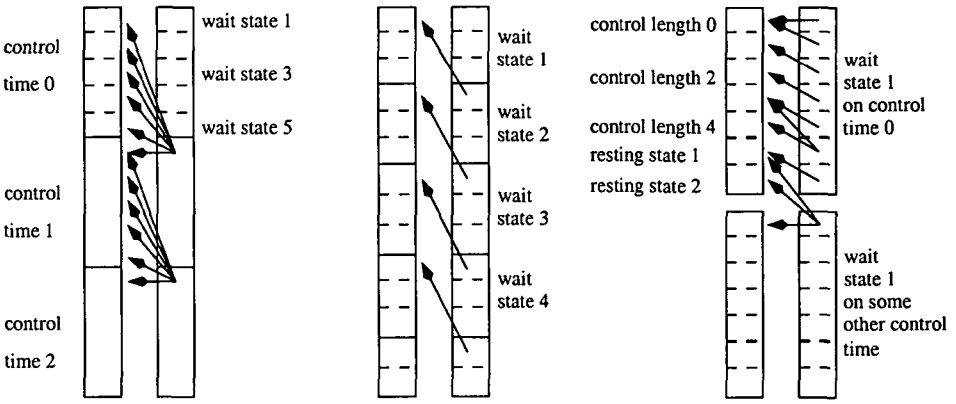

Figure 2: The structure of the state space. Alternative states are similar to control times, except that they do not choose the best control plan available.

When

$$\begin{aligned} &C'^t = C^t + 1, \quad W' = C'^l = 0, \quad 0 \le W \le h^l - 2, \\ &\text{and} \quad C^r(C^m) + C^M - 1 \le C^l \le C^r(C^M) + C^M - 1, \end{aligned} \tag{14}$$

the control at stage k and in state (C^t, W, C^l) has stopped, which in turn increases the amount of control times by one ($C'^t = C^t + 1$). The next control starts in state $(C'^t, 0, 0)$ at stage $k+1$. We add $C^M + 1$ both for minimum and maximum resting times, because C^l indexes the control length and the resting time (see Figure 2), and because the states corresponding to the resting time are located next to the control lengths (i.e., after state $C^M - 1$). Note, that we restrict the number of wait states by $h^l - 2$ (wait states can get $h^l - 1$ different values).

Moreover, it is possible that "old" optimal plan at stage k does not change (or be better) when moving into stage $k+1$, and so

$$C'^t = C^t, \quad W' = C'^l = 0, \quad \text{and} \quad W = C^l = 0. \tag{15}$$

This is the only case with conditions (14) and (19), where DP (recursion formula (12)) can make decisions about the path. If two paths give the same result, DP (12) chooses the one with a later control. This does not have any impact on the result, but in practice we usually want to do the controls as late as possible.

Figure 2 shows the state structure. Conditions (14) and (15) are shown on the left. There we have several states, from which we choose the maximum. When

$$C'^t = C^t, \quad W' = W + 1, \quad \text{and} \quad C'^l = C^l = 0, \tag{16}$$

we "move some information from the past" to new stage $k+1$. With this information we can check what result can be achieved, if we choose the best path W stages ago instead of some other control plan with the last control started in the interval

$[k - W, k]$. Figure 2 shows this in the middle, where a control time is depicted. When

$$C'^t = C^t, \quad W' = W, \quad C'^l = C^l + 1, \quad \text{and} \quad (C'^l \neq 1,\ C'^l \neq C^M + 1), \tag{17}$$

we increase the control length (the control started C^l time points ago). When

$$C'^t = C^t \quad W' = W, \quad \text{and} \quad 1 = C'^l = C^l + 1, \tag{18}$$

a new control starts. In this situation we add the cost of control P' to the result. Finally, when

$$C'^t = C^t \quad W' = W, \quad C'^l = C^M + 1, \quad \text{and} \quad C^m \leq C^l \leq C^M, \tag{19}$$

we can calculate the impact of a legal control on a clipping situation. Figure 2 shows conditions (17)–(19) in the right.

For each state (C^t, W, C^l) and for each stage $k > 0$, we save the connection pointing to some state of the previous stage. The connections form a *path*. When we have the values

$$R'(\mathbf{s}, (C^t, W, C^l), N)$$

with appropriate values in C^t, W and C^l, we can form the control plan $\mathbf{C}$ by traversing the path formed by the connections. The path is optimal with respect to the state space used (but not with respect to the problem). Note that functions R' and R'' in equations (11)–(13) comply with the conditions (7)–(9).

4 The properties of the state space

Next we study the properties of the dynamic recursion formula (12)–(13). Consider stage i. A *local* control for state $(C^t + 1, 0, 0)$ is a control formed at control time $C^t + 1$, stopped at stage $j > i + C^r(C^m)$, and using the control plan formed at stage i for state $(C^t, 0, 0)$. Stages $k > i$ do not belong to the local control, provided that we do not use the control plan of state $(C^t, 0, 0)(i)$ at stage k. This means that the wait states are not considered when forming a local control.

In the next theorem we suppose that all references to wait states have been omitted from the conditions (14), (15), (18) and (19).

Theorem 1. *State space (C^t, C^l) finds, for each stage i, the best local control following stage i.*

Proof. Consider the controls starting after stage i from state $(C^t, 0)$ and using control plan $\mathbf{C}$ determined by i and $(C^t, 0)$. The conditions (14) and (15) determine the best control for state $(C^t + 1, 0)(j)$, according to the equation (12). The condition (14) gives the maximum because of the conditions (18) and (19). □

Corollary 1. *State $(1, 0, 0)(N)$ gives the best control plan having one control.*

Note that the length and the amount of payback do not have any consequences in the case of Theorem 1. State space (C^t, C^l) gives sub-optimal results [1], which can be improved with wait states (still being sub-optimal).

Let $\mathbf{C}_i, \mathbf{C}_{i+1}, \ldots, \mathbf{C}_a$ be the control plans of the control time C^t and the stages (time points) $i, i+1, \ldots, a$, respectively, i being the first time point of an hour. In the next theorem we show that it is enough to choose between the control plans $\mathbf{C}_i, \mathbf{C}_{i+1}, \ldots, \mathbf{C}_a$, when forming a control $[a, b]$. This refers to the situation in condition (14) of DP (12), where we check, how well the control plans of states $(C^t, 0, 0)(i), (C^t, 0, 0)(i+1), \ldots, (C^t, 0, 0)(a)$ work with the control starting at time point a.

Intuitively, the next theorem is based on the property that if the last control of some control plan stops with resting time in the "previous hour", it will not have any impact on the controls in the "present hour".

Theorem 2. *Suppose there is no payback. Suppose further that we start a new control from stage a for control time C^t, which will stop at stage b locally maximizing the control plan $\mathbf{C}_a$. Let i be the first moment of the hour containing a. Now it is enough to choose (with the wait states) from the set of control plans $\mathbf{C}_i, \mathbf{C}_{i+1}, \ldots, \mathbf{C}_a$, when forming a new control $[a, b]$.*

Proof. We show that it is not necessary to reach time points earlier than the start of the present hour. Consider situations where it is possible to choose between control plan $\mathbf{C}_{i-n}$ of time point $i-n$ $(n \geq 1)$, and control plans $\mathbf{C}_i, \mathbf{C}_{i+1}, \ldots, \mathbf{C}_a$, when forming $[a, b]$. Since DP (12) chooses always the maximum, the result of $\mathbf{C}_j$ does not decrease when j increases. Thus, the result of $\mathbf{C}_i$ is at least that of $\mathbf{C}_{i-n}$. If we choose some of the control plans $\mathbf{C}_{i-n}, \ldots, \mathbf{C}_{i-1}$ to be used with a control that starts at a, we obtain at least as good result with the control plan $\mathbf{C}_i$. □

Note that the absence of payback and the fact that the resting time is coded into the state space are crucial here. It follows from Theorem 2 that we need one wait state at the first time point of an hour, two at the second time point and finally $h^l - 1$ at the last time point of an hour (h^l is the length of an hour). In other words, we need on the average $(h^l - 1)/2$ wait states at each time point. (In [1] we used $h^l - 1$ wait states at each time point.)

In general, the state space (C^t, W, C^l) does not achieve the optimal result when the length of payback is non-zero (a sample case is given in Section 5). We need at least one more state variable to be able to form a better path [2, pp. 30–34]. With variable A we check the non-maximum paths according to (12) for the three state variable system.

A *local alternative* of stage i is a control which stops at the stage i including the resting time and which is not chosen into the control plan. A three variable system chooses the best alternative among several, as shown in the left side of Figure 2. We set this to be the alternative state one. In the alternative state two, we choose the second best path from the control time C^t for the first state of control time $C^t + 1$. The third alternative state uses the third best path found so far and so on.

Alternative states can be easily implemented. When looking for the best path, we can cater the required amount of paths to find the Ath path. Moreover, the solution (a path) given by the condition (15) has to be checked when we are looking for the number of the best paths. The starting configuration (13) and the transition conditions (14)–(19) work with alternative states without major modifications. The initial solution is calculated only for the first alternative state and the conditions (14)–(19) work inside an alternative state just as in the case of the three variable system.

5 Tests

We first checked that Theorem 2 gives twice as fast method as the old DP. After that, we wanted to see whether paybacks affect the solutions given by the new three state variable DP. When we saw that paybacks do affect the results, we tried to improve the accuracy by increasing the number of wait states.

Intuitively, the wait states starting at point a can only "look up" that far (to point a) later on at the moment b. So if a is the beginning of an hour, we "do not see" into the previous hour at moment b and cannot make a choice between earlier control plans. When there is no payback, the number of wait states equaling to the number of time points after a start of an hour is enough, because a control finished in the previous hour does not affect the present hour to be cut. However, when we are using payback, we need to be able to look further into the previous hours in order to increase the accuracy. By adding c wait states, we directly reach earlier moments. We are tempted to think that increasing c will improve the solution. Our tests, however, show that while this is mostly true, there are exceptions.

In the tests we used a group shown in the upper left corner in Figure 3. We tested the group with 5 different clipping situations. Tests 1–4 could be clipping situations occurring in reality. They are similarly shaped and contain "morning and afternoon" consumption peaks. The shapes are at different levels giving tests

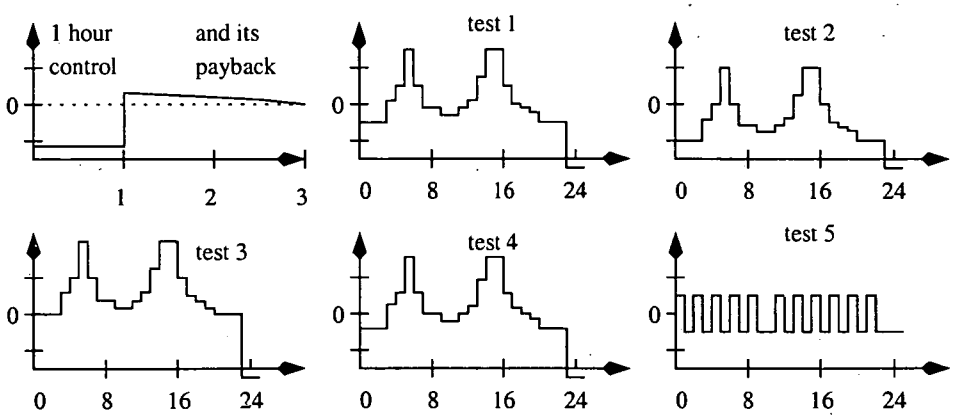

Figure 3: Payback used in the tests and the test cases.

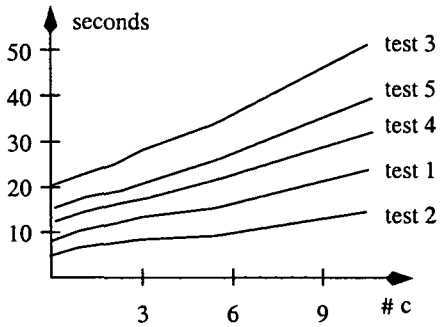

Figure 4: Running times.

of different difficulty. Test 5 is artificial. When load curve is above 0, we have overload that should be cut off. One tick stands for 1 MW. (Horizontal axis are for time.)

Each hour is discretized into twelve time points. The group had 1.2 MW control capacity (i.e., 0.1 for five minutes). The payback is two hours long, the minimum control length is 30 minutes and the maximum control length is one hour. One MW overload costs −99 000 and underload −900. Resting time is 10 minutes.

Figure 4 contains running times (in seconds) for the old DP solution and for DPs with $c = 0, 1, 2, 3, 5$ and 11 (horizontal axis). Moreover, $h^l = 12$. We see that the running times increase almost linearly on the number of waits states. Table 2 contains the results for the tests used in Figure 3. Here we see that, in general, the results improve if the number of wait states is increased.

In the second test, however, we see that increasing the number of wait states has decreased the result between DPs with $c = 3$ and $c = 5$ (a locally better solution is worse). This somewhat non-intuitive result follows from the fact that our DP solutions do not fulfill the optimality principle usually stated for dynamic programming solutions [2, p. 16]. Reason for this is that we cannot guarantee that optimal solution at stage i entails optimal solution for the rest of the case. Payback may affect later hours and decisions. This information should be available at the moment when we are deciding the length of a control.

We did not use any alternative states in the test series reported in Table 2 and

Table 2: Solutions without alternative states.

Test	old DP	$c = 0$	$c = 1$	$c = 3$	$c = 5$	$c = 11$
1.	−144 478	−144 478	−144 478	−144 478	−144 478	−144 478
2.	−15 603	−15 662	−15 662	**−15 662**	**−15 778**	−15 603
3.	−368 665	−371 296	−370 238	−368 665	−368 665	−368 665
4.	−174 490	−175 668	−175 668	−175 668	−175 668	−174 490
5.	−9 860	−9 860	−9 860	−9 860	−9 860	−9 860

Table 3: The best solutions with alternative states.

Test	old DP	$c = 0$	$c = 1$	$c = 3$	$c = 5$	$c = 11$
1.	**−144 478**	−144 478	−144 478	−144 478	−144 478	−144 478
2.	−15 603	−15 662	−15 662	**−15 487**	**−15 487**	**−15 487**
3.	−368 373	−369 069	**−368 046**	−368 162	−368 278	−368 373
4.	−174 490	−175 668	−175 543	**−174 398**	**−174 398**	−174 490
5.	−8 956	−8 336	−8 351	−8 186	−7 742	**−7 538**

Figure 4. We run the same test series using 2, 5, 10 and 15 alternative states. Table 3 contains the best solutions found among all the test series. Results were improved in most of the cases, sometimes over 10%.

Test 5 informatively demonstrates how results improve as the number of wait states or alternative states (or both) increases. The results and running times are shown in Figures 5 and 6. We also tried DPs with 30 and 100 alternative states. DP with 15 alternative states gives −8 336, with 30 states −7 872, and with 100 states −7 214, which is better than the solution given by 15 alternative and 11 additional wait states (see Table 3). Our conclusion is that a clipping situation with many overload intervals most likely benefits from the use of alternative states.

We also studied how alternative states and wait states together improve the results and how they affect the running times. The left hand side of Figure 5 contains the results for different alternative state amounts (1, 2, 5, 10 and 15). As the number of wait states is increased, the results improve in general. There are exceptions where few wait states do worse than DP with $c = 0$ (see lines for A5 and A15). The number of additional wait states used is irrelevant for this instance, when we used only one alternative state.

On the right side the same data is plotted for five different additional wait state amounts as well as for the old DP system. We conclude that the number of alternative states is much more crucial for the results than the number of wait states. Alternative states also improve the results of DP system with fixed amount

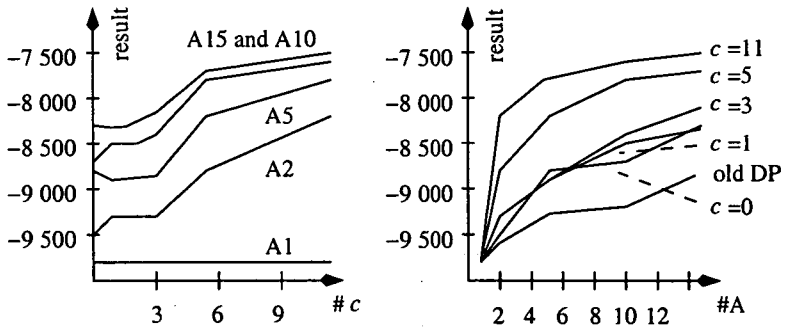

Figure 5: Wait and alternative states, results for test 5.

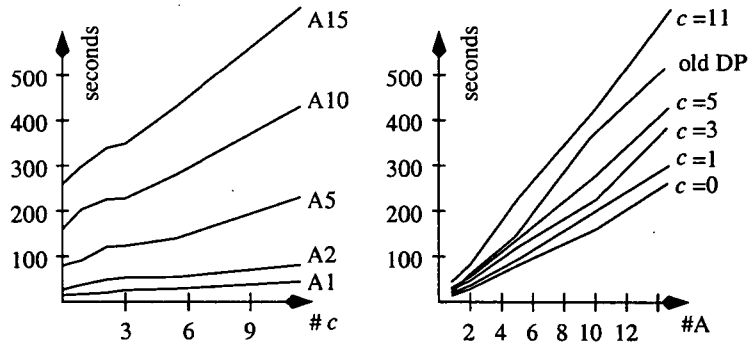

Figure 6: Wait and alternative states, running times for test 5.

of wait states (in old DP $c = h^l - 1$).

Figure 6 contains the running times for test 5. Execution decelerates almost linearly as the number of alternative or wait states is increased. The number of wait states in one hour is $\sum_{a=0}^{h^l-1} (c + (a \mod h^l))$ (where $c = 0, \ldots, h^l - 1$ and a is a moment). Hence, the increase of c by one gives $h^l - 1$ additional wait states for one hour. This is proportionally less than the increase brought by the increase of the number of alternative states by one, which is the number of used wait states in one hour. This explains why the increase of the number of alternative states decelerates more the running times than the increase of the number of wait states.

6 Conclusions

In this work we have analyzed the properties of the state space of a dynamic programming problem arising in direct load control, and quicker optimization algorithms are formed without sacrificing the accuracy of the results when payback is not used. Moreover, we have found practical ways to improve the results by increasing the state space when payback is used. We have described (sub-optimal) solutions for three and four state variables. If the result accuracy is not crucial, one can drop wait states away, arriving to a faster two state variable solution of [8].

There are still open problems concerning the properties of state variable C^t. They seem to behave in a way that enables us to reduce the number of states used (for details, see [1]). Moreover, we conjecture that the control length C^l has properties, by which we can further speed up the algorithms.

If there is enough time, it is possible to add a new state variable, called alternative state. With four state variables we achieve even better results, as is shown in our tests. Most of the time, additional wait states as well as additional alternative states improve the results. Hence, one can choose between fast inaccurate, and accurate but slow solutions. Similar trade can be made between two, three and four variable state spaces. The alternative states seem to improve the results also in the cases occurring in production systems.

Acknowledgment

The author is grateful to Erkki Mäkinen for his time and comments and to an anonymous referee whose valuable comments improved the quality of this work.

References

[1] Isto Aho, Harri Klapuri, Jukka Saarinen, and Erkki Mäkinen. Optimal load clipping with time of use rates. *International Journal of Electrical Power & Energy Systems*, 20(4):269–280, May 1998.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 1996.

[3] R. Bhatnagar, J. Latimer, L.J. Hamant, A.A. Garcia, G. Gregg, and E. Chan. On-line load control dispatch at Florida Power & Light. *IEEE Transactions on Power Systems*, 3(3):1237–1243, August 1988.

[4] R. Bhatnagar and S. Rahman. Dispatch of direct load control for fuel cost minimization. *IEEE Transactions on Power Systems*, PWRS-1(4):96–102, November 1986.

[5] K. Bhattacharyya and M.L. Crow. A fuzzy logic based approach to direct load control. *IEEE Transactions on Power Systems*, 11(2):708–714, May 1996.

[6] Douglas W. Caves and Joseph A. Herriges. Optimal dispatch of interruptible and curtailable service options. *Operations Research*, 40(1):104–112, January–February 1992.

[7] Wen-Chen Chu, Bin-Kwie Chen, and Chun-Kuei Fu. Scheduling of direct load control to minimize load reduction for a utility suffering from generation shortage. *IEEE Transactions on Power Systems*, 8(4):1525–1530, November 1993.

[8] Arthur I. Cohen and Connie C. Wang. An optimization method for load management scheduling. *IEEE Transactions on Power Systems*, 3(2):612–618, May 1988.

[9] I.M. El-Amin, A.R. Al-Ali, and M.A. Suhail. Direct load control using a programmable logic controller. *Electric Power Systems Research*, 52(3):211–216, 1999.

[10] A.J. Gaul, E. Handschin, W. Hoffmann, and C. Lehmköster. Establishing a rule base for a hybrid es/xps approach to load management. *IEEE Transactions on Power Systems*, 13(1):86–93, February 1998.

[11] Yuan-Yih Hsu and Chung-Ching Su. Dispatch of direct load control using dynamic programming. *IEEE Transactions on Power Systems*, 6(3):1056–1061, August 1991.

[12] C.N. Kurucz, D. Brandt, and S. Sim. A linear programming model for reducing system peak through customer load control programs. *IEEE Transactions on Power Systems*, 11(4):1817–1824, November 1996.

[13] Jean-Charles Laurent, Guy Desaulniers, Roland P. Malhamé, and François Soumis. A column generation method for optimal load management via control of electric water heaters. *IEEE Transactions on Power Systems*, 10(3):1389–1400, August 1995.

[14] Kah-Hoe Ng and Gerald B. Sheblé. Direct load control — a profit-based load management using linear programming. *IEEE Transactions on Power Systems*, 13(2):688–695, May 1998.

[15] B. Rautenbach and I.E. Lane. The multi-objective controller: a novel approach to domestic hot water load control. *IEEE Transactions on Power Systems*, 11(4):1832–1837, November 1996.

[16] Paresh Rupanagunta, Martin L. Baughman, and Jerold W. Jones. Scheduling of cool storage using non-linear programming techniques. *IEEE Transactions on Power Systems*, 10(3):1279–1285, August 1995.

[17] Katsuyuki Tomiyama, John P. Daniel, and Satoru Ihara. Modeling air conditioner load for power system studies. *IEEE Transactions on Power Systems*, 13(2):414–421, May 1998.

[18] Deh-chang Wei and Nanming Chen. Air conditioner direct load control by multi-pass dynamic programming. *IEEE Transactions on Power Systems*, 10(1):307–313, February 1995.

[19] Allen J. Wood and Bruce F. Wollenberg. *Power Generation, Operation and Control.* John Wiley & Sons, 1984.

Received November, 2002

Acta Cybernetica **16** (2004) 443–449.

On Computing the Hamming Distance

Gerzson Kéri* and Ákos Kisvölcsey†

Abstract

Methods for the fast computation of Hamming distance developed for the case of large number of pairs of words are presented and discussed in the paper. The connection of this subject to some questions about intersecting sets and Hadamard designs is also considered.

Keywords: covering radius, Hamming distance, Hamming weight, intersecting sets, minimum distance.

1 Introduction and notation

Let Z_q^n denote the set of all n-tuples $(x_1, x_2, \ldots, x_n)$, where $Z_q = \{0, 1, \ldots, q-1\}$. The elements of the set Z_q^n are called words, and the Hamming distance $d(x, y)$ between two words $x, y \in Z_q^n$ is defined as the number of coordinates in which they differ.

One may encounter the problem of determining the Hamming distance for a large number of pairs of words in the same space. This is, for example, the case when the minimum distance or the covering radius for a lot of codes $C_i \subseteq Z_q^n$ are to be determined. (See also Section 6.) The Hamming distance and Hamming weight find many applications also in cryptography [5]. For problems like this there emerges the need for faster computation.

In the paper a general method is presented and discussed for the fast computation of the Hamming distance. This method is related to a problem of intersecting sets.

We emphasize that the suggested (and applied) method is not faster than the direct method if the Hamming distance is to be determined for only a small number of pairs of words. It is proposed for application only if the number of pairs is large enough.

The notation & is used for the bitwise "and" operation, XOR for the bitwise "exclusive or" operation. The wgt function counts the number of 1-s in a binary

*Computer and Automation Research Institute, Hungarian Academy of Sciences, H-1111 Budapest Kende u. 13-17, Hungary, e-mail: keri@sztaki.hu

Supported in part by the Hungarian National Research Fund OTKA, Grant No. T043276.

†Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences, H-1053 Budapest Reáltanoda u. 13–15, Hungary, e-mail: ksvlcs@renyi.hu

integer; it can be given by formula as

$$\mathrm{wgt}(a) = \sum_{k=0}^{\infty} \left(\lfloor a/2^k \rfloor \pmod 2 \right).$$

The symmetric difference of two sets is denoted by $\triangle$:

$$A \triangle B = (A \cap \overline{B}) \cup (\overline{A} \cap B).$$

2 Hamming distance of q-ary vectors and q-ary distance of integers

Clearly, there is a one-to-one correspondence between a word $x = (x_1, x_2, \ldots, x_s) \in Z_q^s$ and a nonnegative integer n in the interval $0 \leq n \leq q^s - 1$:

$$x \longleftrightarrow n = \sum_{i=1}^{s} x_i q^{s-i}.$$

We define the q-ary distance $d_q(a, b)$ of two nonnegative integers as the Hamming distance of the corresponding words in any space Z_q^s where

$$s \geq \max\left(\log_q(a+1), \log_q(b+1)\right).$$

We look for a fast way of computing the Hamming distance of words, stored in the form of q-ary integers for a large number of pairs of words in the same space. That means the computing of $d_q(a, b)$ for pairs of integers (a, b). This problem arises, for example, when the minimum distance or the covering radius of many codes are to be checked.

The minimum distance of a code $C \subseteq Z_q^s$ is defined as

$$\min\{d(x, y) \mid x, y \in C,\ x \neq y\}.$$

The covering radius of a code $C \subseteq Z_q^s$ is the smallest positive integer R such that for an arbitrary $x \in Z_q^s$, there exists one (or more) $y \in C$ with $d(x, y) \leq R$. In other words,

$$R = \max\{d(x, C) \mid x \in Z_q^s\},$$

where

$$d(x, C) = \min\{d(x, y) \mid y \in C\}.$$

3 The binary case ($q = 2$)

Fast methods to calculate Hamming distances (and Hamming weights) in the binary case are known from the literature, see e.g. [5] where the theme is discussed within a more general context. There can be found many communications as well as computer codes related to the subject also on the web.

Here, we describe in short the substance of the method as follows.

For $q = 2$, i. e. for binary numbers, clearly

$$d_2(a, b) = \mathrm{wgt}(a \text{ XOR } b).$$

This fact suggests arranging the weights into an array consisting of the array elements

$$\mathrm{wgt}(1), \quad \mathrm{wgt}(2), \quad \ldots, \quad \mathrm{wgt}(2^L - 1),$$

where the exponent L depends on the computational environment (available hardware and software, programming language etc.).

The same method can be applied with a slight modification also for numbers greater than $2^L - 1$ if we split them into 2 or more parts. If, e.g., $n > 2^L - 1$ but $n \leq 2^{2L} - 1$, then – referring to the identity

$$\mathrm{wgt}(n) = \mathrm{wgt}\left(\lfloor n/(2^L) \rfloor\right) + \mathrm{wgt}\left(n \,(\mathrm{mod}\ 2^L)\right),$$

– we can use the formula

$$d_2(a, b) = \mathrm{wgt}\left(\lfloor (a \text{ XOR } b)/(2^L) \rfloor\right) + \mathrm{wgt}\left((a \text{ XOR } b) \,(\mathrm{mod}\ 2^L)\right).$$

That way, an array of length 2^L is enough for treating integers as large as we want.

Note that the division by 2^L can be performed simply by a right shift of the dividend.

4 Method for the case $q > 2$

When $q > 2$, the q-ary distance $d_q(a, b)$ of two integers cannot be determined immediately by the help of the weight function. What can be done is to have a and b mapped to (longer) integers A and B such that

$$d_2(A, B) = k \cdot d_q(a, b) \quad \text{for any } a, b \in Z_q^s,$$

where k is a positive integer, depending only on the value of q and the mapping.

For this purpose, let

$$\varphi_q : Z_q \longrightarrow Z_2^t$$

with an appropriate t, a mapping having the property of

$$\mathrm{wgt}\left(\varphi_q(\alpha) \text{ XOR } \varphi_q(\beta)\right) = k \tag{1}$$

for any pair $\alpha, \beta \in Z_q$, $\alpha \neq \beta$ for a positive integer k.

Clearly, φ_q generates a mapping of Z_q^s to Z_2^{st}, if we apply φ_q to all q-ary digits of $n \leq q^L - 1$. The corresponding mapping for q-ary integers can be written by the formula

$$\Phi_q(n) = \sum_{j=0}^{L-1} 2^{jt} \cdot \varphi_q\left(\lfloor n/q^j \rfloor \pmod q\right).$$

Now, for any $a, b \leq q^L - 1$, $\operatorname{wgt}(\varphi_q(\alpha) \text{ XOR } \varphi_q(\beta)) = k$ implies $\operatorname{wgt}(\Phi_q(a) \text{ XOR } \Phi_q(b)) = k \cdot d_q(a, b)$.

From the point of view of effectiveness, the value of t should be kept as small as possible.

The same problem can be translated to a problem with intersecting sets. For this purpose, consider a set S consisting of t elements:

$$S = \{u_1, u_2, \ldots, u_t\}.$$

Consider also the binary representation of $\varphi_q(\alpha)$ as

$$\varphi_q(\alpha) = (b_1(\alpha), b_2(\alpha), \ldots, b_t(\alpha))$$

for any $\alpha \in Z_q$, $\varphi_q(\alpha) : Z_q \longrightarrow Z_2^t$.

Define the subsets $S_1, S_2, \ldots, S_q$ of S as follows:

$$u_i \in S_{\alpha+1} \text{ if and only if } b_i(\alpha) = 1.$$

To find a mapping $\varphi_q(\alpha)$ having the property (1) is equivalent to find a set S and q subsets $S_1, S_2, \ldots, S_q \subseteq S$ such that the cardinality of the symmetric differences

$$S_i \triangle S_j = (S_i \cap \overline{S_j}) \cup (\overline{S_i} \cap S_j)$$

is constant for any pairs of S_i and S_j, provided $i \neq j$, where $\overline{S_i}$ is used for $S \setminus S_i$ $(i = 1, 2, \ldots, q)$.

For the system of sets $S_1, S_2, \ldots, S_q$ with the property described above, the following notices can be taken.

1. Consider the sets

$$U_i = S_1 \triangle S_{i+1}$$

for $i = 0, \ldots, q-1$. Now, we have $U_0 = \emptyset$, and

$$|U_i| = k$$

for $i = 1, \ldots, q-1$. It is easy to see that $U_i \triangle U_j = S_i \triangle S_j$, thus also $|U_i \triangle U_j| = k$ holds. Clearly, $|U_i \triangle U_j| = |U_i| + |U_j| - 2|U_i \cap U_j|$, consequently,

$$|U_i \cap U_j| = \frac{k}{2}$$

for every $i, j \geq 1$, $i \neq j$. From this, it also follows that k must be even. So, we have a k-uniform family $U_1, \ldots, U_{q-1}$ on the t-element ground set S, such that any pair

of sets shares the same number of elements. By using linear algebraic methods, Bose [2] proved that $t \geq q-1$ for such set-systems. Later in the paper we show that this bound can be achieved in some cases (cf. Examples 2, 3).

2. Assume now that $t = q-1$. Ryser [7] showed that in this case every point in S is contained in exactly k sets from $U_1, \ldots, U_{q-1}$. By doubly counting the triplets (u, U_i, U_j), where $u \in U_i \cap U_j$, $i \neq j$, we get

$$t\binom{k}{2} = \binom{q-1}{2}\frac{k}{2}.$$

From this, we obtain $q = 2k$. Since k is even, if q is not divisible by 4, then $t \geq q$ must hold. Obviously, this bound can be achieved in any case (cf. Example 1).

3. Suppose that q is divisible by 4. Let $q = 4\lambda$, $k = 2\lambda$, where λ is a positive integer. What we want to find is a symmetric block design $S_\lambda(2, 2\lambda, 4\lambda - 1)$, that is, a 2λ-uniform set-system $U_1, \ldots, U_{4\lambda-1}$ on a $t = 4\lambda - 1$-element ground set S, such that every pair of sets has an intersection of size λ. If we take the complement sets $V_i = S \setminus U_i$, then

$$|V_i| = 2\lambda - 1,$$

and $V_i \cap V_j = S \setminus (U_i \cup U_j)$. Since $|U_i \cup U_j| = |U_i| + |U_j| - |U_i \cap U_j| = 3\lambda$, we have

$$|V_i \cap V_j| = \lambda - 1.$$

So, equivalently, we want to find a so-called *Hadamard design* $S_{\lambda-1}(2, 2\lambda-1, 4\lambda-1)$. It is known that such a system exists if and only if there is a Hadamard matrix of order 4λ. An *Hadamard matrix of order* m is an $m \times m$ matrix H with entries $\{1, -1\}$ such that its row vectors are orthogonal to each other, as well as its column vectors, i.e., $HH^T = H^T H = mI$. It is conjectured that there is an Hadamard matrix of order 4λ for every positive integer λ, and thus, we can have $t = q - 1$.

5 Examples

1. For arbitrary $q > 2$, we may choose $t = q$ and $\varphi_q(\alpha) = 2^\alpha$.
   Then, $\mathrm{wgt}\,(\varphi_q(\alpha)\ \mathrm{XOR}\ \varphi_q(\beta)) = 2$ for $\alpha \neq \beta$.
   In the terminology of intersecting sets

$$S = \{u_1, u_2, u_3\},\ S_1 = \{u_1\},\ S_2 = \{u_2\},\ S_3 = \{u_3\}.$$

2. For $q = 4$, let $t = 3$ and $\varphi_4(\alpha) = 0, 3, 5, 6$ for $\alpha = 0, 1, 2, 3$, respectively.
   Now, $\mathrm{wgt}\,(\varphi_4(\alpha)\ \mathrm{XOR}\ \varphi_4(\beta)) = 2$ again for $\alpha \neq \beta$.
   In the terminology of intersecting sets

$$S = \{u_1, u_2, u_3\},\ S_1 = \emptyset,\ S_2 = \{u_1, u_2\},\ S_3 = \{u_1, u_3\},\ S_4 = \{u_2, u_3\}.$$

3. For $q = 2^{m+1}$, $m \geq 1$, the following recursion can be applied:

$$\varphi_{2^{m+1}}(2\alpha - 1) = (2^{2^m} + 1) \cdot \varphi_{2^m}(\alpha),$$

$$\varphi_{2^{m+1}}(2\alpha) = (2^{2^m} - 1) \cdot (\varphi_{2^m}(\alpha) + 1) .$$

In this case $t = q - 1 = 2^{m+1} - 1$ can be specified. The inequality

$$\varphi_{2^{m+1}}(\alpha) \leq 2^{2^{m+1}-1} - 1 \quad \text{for} \quad 0 \leq \alpha \leq 2^{m+1} - 1$$

can be proved by induction. The multiplier k assumes the value 2^m.

6 Application of the method for checking the covering radius of codes

The methods described in the paper found an application in [4] for computing the covering radii of a huge number of codes. This computation resulted in the improvement of known lower bounds on the covering radii for several families of codes. This way, general inequalities (sometimes equalities) were found for the covering radii of an infinite number of codes; however, to obtain these results, a finite (but very large) number of codes had to be considered and the covering radii of more than 150 million codes were checked by using a computer.

This job could not have been completed within a reasonable time by applying the direct method for the computation of the Hamming distance, i. e. by counting the number of non-identical coordinates.

By using the weight function and the "exclusive or" operation, the check of binary codes was completed 6–8 times faster than by the direct method. For ternary and mixed ternary/binary codes, using the mapping φ and applying the weight function for the transformed vectors resulted in an additional gain in the CPU time. Thus, finally, the whole job of checking the covering radii of millions of codes required about 30 days of CPU time (instead of 300 days or more, which would have been required by applying the direct method).

Finally, we summarize the computational aspects of the method applied for the case of a mixed ternary/binary Hamming space. The process of the method needs three initial steps as follows:

1. We start with storing in two arrays the powers of 2 and 3 for exponents $0, 1, \ldots$ until these can be represented as long integers (arrays pow2 and pow3).

2. The weights of binary integers are stored in another array wgt of long integers:

$$\mathrm{wgt}(n) = \sum_{j \geq 0} \mathrm{sign}\,(n \;\&\; \mathrm{pow2}(j)) .$$

3. The values of $\Phi_3(n)$ are stored also in an array of long integers:

$$\Phi_3(n) = \sum_{k=0}^{L-1} 2^{3k + (\lfloor n/3^k \rfloor) \,(\mathrm{mod}\ 3)} .$$

After these steps of initialization, the computation of Hamming distances is done as follows.

For arbitrary words x, y of the mixed Hamming space $Z_3^{n_1} \oplus Z_2^{n_2}$, these words can be given as pairs consisting of a ternary and a binary integer:

$$x = (x_t,\ x_b),\ y = (y_t,\ y_b)..$$

Then, the Hamming distance $d_{3,2}(x, y)$ is computed by using the formula

$$d_{3,2}(x, y) = \frac{\mathrm{wgt}(\Phi(x_t)\ \mathrm{XOR}\ \Phi(y_t))}{2} + \mathrm{wgt}(x_b\ \mathrm{XOR}\ y_b).$$

Acknowledgement

The authors are grateful to Patric R. J. Östergård for his helpful comments and suggestions. The first author would like to thank the Hungarian National Research Fund (OTKA) for partial financial support (Grant No. T043276).

References

[1] I. Anderson, *Combinatorics of finite sets*, The Clarendon Press, Oxford University Press, New York (1987).

[2] R. C. Bose, A note on Fisher's inequality for balanced incomplete block designs, *Ann. Math. Statistics*, Vol. 20 (1949) 619–620.

[3] G. Cohen, I. Honkala, S. Litsyn and A. Lobstein, *Covering Codes*, North-Holland, Amsterdam (1997).

[4] G. Kéri and P. R. J. Östergård, Further results on the covering radius of small codes, submitted for publication.

[5] H. Lipmaa and S. Moriai, Efficient Algorithms for Computing Differential Properties of Addition, *Fast Software Encryption '2001 (M. Matsui, ed.), Lecture Notes in Computer Science* Vol 2355., Springer-Verlag (2002), 336–350.

[6] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam (1977).

[7] H. J. Ryser, A note on a combinatorial problem, *Proc. Amer. Math. Soc*, Vol. 1 (1950) 422–424.

Received January, 2004

Acta Cybernetica **16** (2004) 451–458.

The Recycled Kaplansky's Game*

András Pluhár†

Abstract

Motivated by the Nine Men Morris game, the achievement or hypergraph games can be prolonged in the following way. After placing a prescribed number of stones, the players pick some of these up and replace again. We study the effect of this recycling for the k-in-a-row game and some versions of the Kaplansky's game.

Keywords: positional games, achievement games, hypergraphs, Kaplansky.

1 Introduction and Results

A large number of combinatorial games were created from the earliest civilizations up to now; the authors of [7] try the impossible task of introducing a fraction of these. In a fascinating class of those, two players, I and II (later on M and B), put marks or move pieces on a board, while the outcome of the game depends on achieving certain geometrical configurations. The most prominent examples are the ageless Tic-Tac-Toe, the Nine Men's Morris, the Go-moku or its western variant, the 5-in-a-row.

Plenty of interesting games are relatively young, such as the Hex, Bridgit, Shannon's switching game or the Hales-Jewett games. In the case of the so-called *positional* or *achievement games* the rules can be unified. Given a finite or infinite set X (the "board"), the players alternately take elements of X (by marking or putting pieces onto it physically), and there is a fixed $H \subset 2^X$, the *winning sets*. A player wins by taking all the elements of a winning set first. For this sub-class we have a rich and beautiful theory.

Sometimes the players take p and q elements of X in turns, respectively. If $p \neq q$, it is a *biased* game, otherwise it is called *accelerated*, see [4, 5, 6, 10, 11, 12]. Since I always wins or the game is a draw when $p = q$ (see [7]), it also interesting to consider the *strong* or *Maker-Breaker version* of a game. Here Maker (I) wins by occupying a winning set, while Breaker (II) wins not by occupying such a set, but preventing Maker of doing so.

*The author was partially supported by the OTKA Fund T34475.

†Department of Computer Science, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary, e-mail: pluhar@inf.u-szeged.hu

However, this pattern does not fit for such games as the recently solved Connect-4 or Nine Men's Morris, see [1, 2]. In the first case the available moves are restricted, while the whole static approach of the positional games is abandoned in the second. We shall address the issue of the second one and make an attempt to capture the idea of movements for a game. For an arbitrary positional game let us define the rules of the *recycled versions* as follows. For a natural number n the players make the first n steps as before; this is the *first phase*. Then, in the *second phase*, they just make moves with some of their earlier placed pieces in turns, instead of introducing new ones.

In order to investigate the effect of recycling, let us define some games. The first is the well-known *k-in-a-row* game ($k \in N$), which is played by the two players on the infinite (chess)board, or graph paper. They alternately put their own marks to previously unmarked squares, and whoever gets k-consecutive marks first (horizontally, vertically or diagonally) of his own, wins.

An interesting way to alter the *k-in-a-row* game is to relax the consecutiveness condition. We shall call the game $L_k(p, 1; n)$ (or *line game* for short) for which:

1. I and II mark p and 1 squares in every step, respectively.
2. I wins upon getting k, not necessary consecutive, marks in a line (horizontally, vertically or diagonally), which is free of II's marks.
3. the game terminates after n steps.

Then let $RL_k(p, 1; n)$ be the recycled version of $L_k(p, 1; n)$.

Our third subject is the Kaplansky's game, where the players put their marks on the Euclidean plane. Here I wins achieving k marks on a line, provided II has no mark on that line. Now $K_k(p, q)$ stands for the version in which I and II marks p and q points, respectively. Let $K_k(p, q; n)$ be the version which ends after n round, and $RK_k(p, q; n)$ be its recycled version.

Before stating our theorems, let us recall some earlier results on these games.

The *recycled k-in-a-row* (no matter when does the second phase start) turns out to be easy, because the decomposition methods utilized in [7] still work, and give the same bounds. That is even the Maker-Breaker version of the recycled k-in-a-row game is a draw if $k \geq 8$.

Bounds for the games $L_k(p, 1; n)$ and $RL_k(1, 1; n)$ are less obvious, we shall prove:

Theorem 1. *In the Maker-Breaker $L_k(p, 1; n)$ game, Breaker wins if $k \geq p \log_2 n + p \log_2 p + 3p$. On the other hand, Maker wins if $p > 1$ and $k \leq c \log_2 n$ for some $c > 0$.*

Theorem 2. *Breaker wins the Maker-Breaker $RL_k(1, 1; n)$ game if $k \geq 32 \log_2 n + 224$.*

In the version of Kleitman and Rothschild (see in [3]) I (II) wins by getting k (l) points of a line while the opponent has none of that line, respectively. They

prove that, given any $k \geq 1$, there is an $l(k)$ such that II has a winning strategy whenever $l \geq l(k)$. Beck in [4] considers little different games; here I wins with k points on a line, II with l, and I may mark p points on each turn, while II only one per turn. Here II wins if $l < ckp\log(p+1)$, for some $c > 0$. He has also shown there exist $C > c > 0$, such that in $K_k(1,1;n)$: Maker wins if $k < c\log_2 n$ and Breaker wins if $k > C\log_2 n$. For its recycled version we have the following result.

Theorem 3. *Breaker wins the Maker-Breaker $RK_k(1,1;n)$ game if $k > cn^{1/3}$.*

2 Proofs

2.1 Weight functions

In the proof of the Theorems 1, 2 and 3 we heavily use the *weight function method*, which was developed in [5] and developed in [6] and [8]. First let us recall some earlier definitions and results.

A pair of (X,H) is called a *hypergraph* if $H \subset 2^X$. If (X,H) is a hypergraph, then a $(p,q,H)-game$ (or simply hypergraph game) is a game in which I selects p and II select q previously unselected elements of X. The first, who takes all elements of an $A \in H$, wins. A (p,q,H)-game has a so-called *Maker-Breaker version* in which I wins taking an edges of the hypergraph any time. One of the most important result on such games is the Erdős-Selfridge theorem; one of its generalization is due to József Beck.

Theorem 4 ([5]). *Breaker wins the $(p,1,H)-game$ if $\sum_{A\in H} 2^{-\frac{|A|}{p}} < \frac{1}{2}$.*

In our cases this theorem cannot be applied directly, since the hypergraphs involved are infinite, and it is not known if Theorem 4 holds for recycled games. The following lemma is also due to Beck (see [5]). We repeat the proof in order to see the properties of the used weight function.

An edge $A \in H$ is *active* if Breaker has not taken any of its elements.

Lemma 1. *Playing a $(p,1,H)$ game, Breaker can assure that no active edge contains more than $p + p\log_2|H|$ elements taken by Maker.*

Proof of Theorem 4. We may assume Maker starts the game. For any $A \in H$ let $A_k(M)$ and $A_k(B)$ be the number of elements in A, after Makers kth move, selected by Maker and Breaker, respectively. Now, for an $A \in H$

$$w_k(A) = \begin{cases} \lambda^{A_k(M)} \text{ if } A_k(B) = 0 \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$

where $\lambda > 0$, and for any $x \in X$ let $w_k(x) = \sum_{x\in A} w_k(A)$. The numbers $w_k(A)$ and $w_k(x)$ are called the *weight* of A and x (in the kth step), respectively. When it does not cause confusion we may suppress the lower index.

Now selecting an element in the kth step Breaker uses the *greedy* algorithm, i.e. chooses an unselected element $y^k \in X$ of maximum weight. Let $x_1^{k+1}, ..., x_p^{k+1}$ be

the elements selected by Maker in the $(k+1)$st step and $w_k = \sum_{A \in H} w_k(A)$ be the *total sum* or *potential*. For $k \geq 0$, following inequality holds for the potential:

$$w_k - w_k(y^k) + (\lambda^p - 1)w_k(y^k) \geq w_{k+1}.$$

Indeed, w_k decreases by $w_k(y^k)$ upon selecting y^k. The elements selected by I in the $(k+1)$st step cause the biggest increase if $w_k(x_l^{k+1})$ is maximal for $1 \leq l \leq p$, and for all A such that $w_k(A) \neq 0$ we have $x_l^{k+1} \in A$ iff $x_m^{k+1} \in A$, $1 \leq l,\ m \leq p$. Since the increase in this case is just $(\lambda^p - 1)w_k(y^k)$, the inequality is proved. Setting $\lambda = 2^{1/p}$, we get $w_k \geq w_{k+1}$, $k \geq 0$, which justifies that w_k is called potential.

Particularly $w_1 \leq (\lambda^p - 1)|H| + |H| \leq 2|H|$. Since $q = 1$ and the elements of H are the same size, the inequality $\sum_{A \in H} 2^{-|A|/p} < 1/2$ leads to the inequality $2|H| < 2^{|A|/p}$. Assume that Maker wins the game in the kth step. This would imply $w_k \geq \lambda^{|A|} = 2^{|A|/p}$, which contradicts the monotonicity of the potential. □

Proof of Lemma 1. Just take the logarithm of the inequality $\lambda^{A_k(M)} = w_k(A) \leq w_k \leq w_1 \leq 2|H|$ that holds for any active edge $A \in H$. □

2.2 Proof of Theorem 1.

Let us recall that a line L means consecutive squares along an infinite line here (horizontally, vertically or diagonally). Now we have infinitely many interacting sets, so the weight function method does not seem to be helpful. The way to overcome the difficulties is to change the definition of the weights. The price of this is that the potential is no longer a decreasing function, but an increasing one. However, we can control the growth, since the game lasts only n steps.

Let H be the set of all lines, and $L_j(M)$ and $L_j(B)$ the number of squares of line L marked by Maker and Breaker after the jth step, respectively. Now the weight function of L at the jth step:

$$w_j(L) = \begin{cases} \lambda^{L_j(M)} \text{ if } L_j(M) \geq 1 \text{ and } L_j(B) = 0 \\ 0 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

where $\lambda = 2^{\frac{1}{p}}$.

For a square q,

$$w_j(q) = \sum_{L \in H, q \in L} w_j(L)$$

is the weight of q, and

$$w_j = \sum_{L \in H} w_j(L)$$

is the *total weight* at the jth step.

Breaker applies the greedy selection. For the weight functions, similarly to the proof of Theorem 4, we have

$$\sum_{L_j(M) \geq 1} w_{j+1}(L_{j+1}) \leq \sum_{L \in H} w_j(L_j).$$

On the other hand, in each step the number of lines whose weight becomes positive is at most $4p$, and the weight of such a line is no more than $\lambda^p = 2$. That is

$$w_{j+1} \leq w_j + 8p$$

holds for $0 \leq j \leq n$, where $w_0 = 0$. That is if the line L is unblocked at step j (i.e. $L_j(B) = 0$) and $L_j = i$ than

$$\lambda^i \leq 8pj \Leftrightarrow i \leq p(\log_2 j + \log_2 p + 3).$$

Since $0 \leq j \leq n$, the first part of Theorem 1 follows.

The second part is fairly standard, we give just the sketch of its proof. In fact, one (say vertical) winning direction is enough. Maker divides the game into phases. For the sake of simplicity we omit to write the integer parts. In the first phase Maker places $n(p-1)/p$ element in a row. Call a column *i-free* if it contains i marks of Maker, but none of Breaker. At the end of the first phase the number of 1-free columns is at least $n((p-1)/p)^2$. In the ith phase Maker uses up $n((p-1/p))^i$ new mark, each is placed to an $i-1$-free column. It is easy to check that Maker can reach the ith phase if $n((p-1)/p)^i \geq 1$, and uses up at most n marks. That is an i-free column appears if $i \leq c \log_2 n$, where c is about $(\log_2 p - \log_2(p-1))^{-1}$. □

2.3 Proof of Theorem 2.

Breaker divides the game into sub-phases. The first sub-phase is the first phase of the game, then a sub-phase consists of n pair of moves. Defining the weight function as before, but $\lambda = \sqrt{2}$, Breaker places every second mark (the *active* marks) according to the greedy strategy and deposits the others arbitrarily, i.e. *in reserve*). It may happen that one of Breakers reserved marks is already on the square q, which is to be occupied by an active mark of Breaker. In that case Breaker places the new mark arbitrarily (sends it into reserve), and the mark on the square q becomes active.

Considering only the effect of Breakers active marks, the game reduces to the game $L_k(2, 1, n)$. That is Lemma 1 applies, and for any line L if $L_j(M) = i$ and $L_j(B) = 0$, then $i \leq 2(\log_2 j + 4)$ if $0 \leq j \leq n$.

In the other sub-phases Breaker plays a fictitious game, and keeps the status of his marks (active or reserved) strictly. The marks of Maker are indexed by the numbers $1, 2, \ldots, n$. At the beginning of a sub-phase Breaker cannot see Makers marks, and in the jth step Makers new mark and the mark indexed by j become visible for Breaker as new moves. (If Maker moved the jth mark, only one mark becomes visible.)

However Breaker responds only in every second step, using the marks from the reserve. (Breaker does nothing in the odd steps. If picking up a mark and putting back to the same place is permitted, it is easy. If it is not, Breaker designates a mark at the very beginning, which is neither active nor reserved, and moves this mark arbitrarily in the odd steps.)

Trying the previous greedy strategy another difficulty arises. Breaker may not occupy the square q of maximum weight because q has been already taken (by one of Makers invisible marks or one of Breakers own reserve). Then, Breaker blocks the lines going through q, using four marks. (See a similar idea in [12].) Now, looking only Makers visible marks, if for a line L, $L_j(M) = i$ and $L_j(B) = 0$ then $i \leq 16(\log_2 j + 7)$, since after at most 16 moves of Makers, Breaker may reply, and Theorem 1 applies.

By the end of a sub-phase Makers all marks become visible, and a line L, which contain more than $16(\log_2 n + 7)$ of them, is blocked by Breakers reserve. Finally, Breaker starts the next sub-phase renaming his marks, the active ones become reserved and vice versa.

Since the active marks control the invisible marks during a sub-phase, if for a line L the sum of visible and invisible marks of Maker on L is i, and L is not blocked (by the active marks or by the reserve), then $i \leq 32(\log_2 n + 7)$. □

2.4 Proof of Theorem 3.

The most natural idea is to mimic the proof of Theorem 2.

Unfortunately it breaks down irreparably at the point where Breaker wants to occupy, or at least block the point q, which is already taken. The problem is that q can be the element of many lines, so Breaker cannot cancel the weight of q by using only constantly many points.

To overcome this difficulty, we change the weight function and give a more sophisticated analysis of it.

Let the weight of a line L after Maker jth move be

$$w_j(L) = \begin{cases} \lambda^{L_j(M)} \text{ if } L_j(M) \geq c_1 n^{1/3} \text{ and } L_j(B) = 0 \\ 0 \qquad\qquad\qquad\qquad\qquad\quad \text{otherwise} \end{cases}$$

where $\lambda = \sqrt{2}$ and $c_1 > 0$ will be specified later.

As before, for a point x, $w_j(x) = \sum_{L \in H, x \in L} w_j(L)$ is the weight of x, and $w_j = \sum_{L \in H} w_j(L)$ is the *total weight* at the jth step.

However, Breaker uses not only the greedy strategy, the recycled point also have to be designated. When Breaker removes a point y, the total weight function may grow. It grows iff there is a line L containing y such that $L_j(M) \geq c_1 n^{1/3}$ and $L_j(B) = 1$. Obviously the number of such points cannot be bigger than the number of lines containing at least $c_1 n^{1/3}$ points of Maker. To estimate this, we need a definition and a theorem of Szemerédi and Trotter.

An *incidence* of a point and a line is a pair (p, L), where p is a point, L is a line, and p lies on L.

Theorem 5 ([14]). *Let I denote the number of incidences of a set on n points and m lines. Then $I \leq c(n + m + (nm)^{2/3})$.*

Let us note that László Székely published a new, more accessible proof of Theorem 5, see in [13].

An easy corollary of Theorem 5 is that there is a constant c_2 such that the number of lines containing at least k points of S is less than $c_2 n^2/k^3$ whenever $k \leq \sqrt{n}$.

That is if $c_1 > c_2^{1/3}$, then the number of lines containing at least $c_1 n^{1/3}$ points of Maker is less than n. It means Breaker can always find a mark y such that its removal does not affect the value of the total weight function. The steps of Maker and Breaker are $x_1, x_2, \ldots x_i$ and $y_1, y_2, \ldots y_i$, respectively.

As before, for the weight function we have

$$w_{j+2} \leq w_j - w_j(y_j) - w_{j+1}(y_{j+1}) + w_j(x_{j+1}) + w_{j+1}(x_{j+1}) + \frac{2}{c_1} n^{2/3} \lambda^{n^{1/3}} \lambda^{n^{1/3}+1}.$$

Here the term $f(n) := \frac{2}{c_1} n^{2/3} \lambda^{n^{1/3}} + \lambda^{n^{1/3}}$ bounds the growth caused by the lines that of weight becoming positive in the jth and $(j+1)$th steps. By the argument of Theorem 4, $w_j(y_j) \geq w_j(x_{j+1}) + w_{j+1}(x_{j+1})$, since $\lambda = \sqrt{2}$. We also have $w_{j+1}(y_{j+1}) > w_{j+1}/n$, since the number of positive weighted lines is less than n, giving

$$w_{j+2} \leq w_j - \frac{w_{j+1}}{n} + f(n).$$

On the other hand, $w_{j+2} \leq w_{j+1} + f(n)$, or equivalently $w_{j+1} \geq w_{j+2} - f(n)$. That is the value of w_{j+2} is bounded, since if $\frac{w_{j+1}}{n} \geq f(n)$, and then we have $w_{j+2} \leq w_j$. From here one gets that $w_{j+2} \leq (n+1)f(n)$. It means that if for a line L, $L_{j+2}(M) = s$ and $L_{j+2}(B) = 0$, then $(n+1)f(n) \geq w_{j+2} \geq \lambda^s$. Taking the logarithm of both sides, $s \leq 2 \log_2 w_{j+2} \leq 2n^{1/3}$, provided n is big enough. □

2.5 Remarks and Open Questions

As we have seen, there is a large gap between the logarithmic lower and $O(n^{1/3})$ upper bound what Maker can achieve in the recycled Kaplansky's game.

Question 1. *Can the upper or lower bounds of Theorem 3 improved?*

Even less is known about recycled hypergraph games in general. It is easy to give example for which Breaker wins the first phase of the game, while Maker wins the recycled version.

Question 2. *Is there a hypergraph game won by Breaker, but Maker wins its recycled version?*

It is also interesting if the Erdős-Selfridge theorem extends to the recycled games.

Question 3. *Is it true if* $\sum_{A \subset H} 2^{-|A|+1} < 1$, *then Breaker wins the recycled version of the* (X, H) *game?*

Acknowledgments. Many thanks to József Beck for the lots of help and encouragement. I would like to thank the useful advices of the unknown referee, too.

References

[1] L. V. Allis, A Knowledge-Based Approach to Connect-Four. The Game is Solved: White Wins. MSc thesis, Vrije Universiteit, The Netherlands, 1988.

[2] R. Gasser, Solving Nine Men's Morris, *Computational Intelligence* **12(1)**, (1996) 24-41.

[3] D.J. Kleitman and B.L. Rothschild, A generalization of Kaplansky's game, *Discrete Math* **2** (1972) 173-178.

[4] J. Beck, On a generalization of Kaplansky's game, *Discrete Math* **42** (1982) 27-35.

[5] J. Beck, On positional games, *J. of Combinatorial Theory Series A* **30** (1981), 117-133.

[6] J. Beck, Van der Waerden and Ramsey games, *Combinatorica* **1** (1981), 103-116

[7] E.R. Berlekamp, J.H. Conway and R.K. Guy, *Winning Ways*, Volume **2**, Academic Press, New York (1982), 667-693.

[8] P. Erdős and J. L. Selfridge, On a combinatorial game, *J. Combinatorial Theory Series B* **14** (1973) 298-301.

[9] A.W. Hales and R.I. Jewett, Regularity and positional games, *Trans. Amer. Math. Soc.* **106** (1963) 222-229; M.R. # 1265.

[10] A. Pluhár, Generalizations of the game k-in-a-row, Rutcor Research Reports 15-94 (1994).

[11] A. Pluhár, Generalized Harary Games, *Acta Cybernetica* **13** (1997) 77-84.

[12] A. Pluhár, The accelerated k-in-a-row, *Theoretical Comp. Sci.* **271** (1-2) (2002) 865-875.

[13] L. A. Székely, Crossing numbers and hard Erdős problems in discrete geometry, *Combin. Probab. Comput.* **6** (1997), no. 3, 353-358.

[14] E. Szemerédi and W. T. Trotter Jr., Extremal problems in discrete geometry, *Combinatorica* **3** (1983), no. 3-4, 381-392.

Received June, 2003

Acta Cybernetica **16** (2004) 459–472.

Distance Functional Dependencies in the Presence of Complex Values

Sebastian Link* and Klaus-Dieter Schewe*

Abstract

Distance functional dependencies (dFDs) have been introduced in the context of the relational data model as a generalisation of error-robust functional dependencies (erFDs). An erFD is a dependency that still holds, if errors are introduced into a relation, which cause the violation of an original functional dependency. A dFD with a distance $d = 2e + 1$ corresponds to an erFD with at most e errors in each tuple. Recently, an axiomatisation of dFDs has been obtained.

Database theory, however, does no longer deal only with flat relations. Modern data models such as the higher-order Entity-Relationship model (HERM), object oriented datamodels (OODM), or the eXtensible Meakup Language (XML) provide constructors for complex values such as finite sets, multisets and lists. In this article, dFDs with complex values are investigated. Based on a generalisation of the HAmming distance for tuples to complex values, which exploits a lattice structure on subattributes, the major achievement is a finite axiomatisation of the new class of dependencies.

Keywords. functional dependencies, complex value data models, error-robustness

1 Introduction

In [3] Demetrovics, Katona and Miklós introduced error-correcting keys in the RDM and generalised them to error-correcting functional dependencies in [4]. In both cases they studied the relationship of these dependencies to inclusion-free sets of attributes and derived combinatorical results on the size of the elements in such families. As these kinds of dependencies provide information about relations that is stable under the introduction of errors, we prefer to talk of *error-robust functional dependencies* (erFDs).

The work on error-robust functional dependencies is motivated by the fact that a database user may be confronted with a relation that contains errors. It is presumed that the user knows the structure of the relation schema, i.e. the attributes and

*Information Science Research Centre, Massey University, Private Bag 11222, Palmerston North, New Zealand, [s.link|k.d.schewe]@massey.ac.nz

the dependencies. There are various reasons for such errors to occur. For instance, a relation may have been transmitted through a noisy channel, so knowing about erFDs may help to localise the errors.

On the other hand, errors may have been introduced deliberately in order to hide and secure data. Then the knowledge about erFDs permits drawing conclusions about the errors. So the study of erFDs may lead to results on how reliable the used data hiding mechanism is. Another reason for an errornous relation may be that the data has been spoiled deliberately. Analogously to the case of using a noisy transmission channel the knowledge about erFDs may help to detect the errors.

In the conclusion of [4, p.92] the authors pose the question, whether erFDs in the RDM can be characterised, i.e., finitely axiomatised. As already shown in [4, Prop. 1.1] erFDs are subsumed by another class of dependencies, called *distance functional dependencies* (dFDs), where the distance refers to the Hamming distance of tuples projected to the left hand side of the dependency. More precisely, an erFD for the case of at most e errors in each tuple corresponds to a dFD with a distance of at most $2e + 1$. A finite axiomatisation of distance functional dependencies for the RDM including the more general case of disjunctive distance functional dependencies was achieved in [8].

Over the last decade the major focus of database theory has shifted from the relational data model to data models with complex values rather than just tuples. Examples are the higher-order Entity-Relationship model (HERM, [11]), object oriented datamodels (OODM, [10]), or most recently the eXtensible Meakup Language (XML, [1]). A natural question is, whether the theory of functional dependencies and distance functional dependencies can be carried over to these data models. For FDs this problem was addressed in [5] for finite sets, then generalised in [7] to capture sets, lists and multisets, and in [6] to capture sets and disjoint unions. In all these cases a finite axiomatisation could be achieved.

The aim of this paper is to generalise the notion and finite axiomatisation of error-robust functional dependencies. In Section 2 we summarise the results from [8] on dFDs in the relational data model, excluding the disjunctions. The Section 3 we introduce vthe fundamentals of nested attributes, which capture the gist of higher-order data models. We present some results from [7] that will be needed in this article. Section 4 introduces distance functional dependencies on nested attributes and a sound set of derivation rules for such dependencies. Finally, the completeness of this set of rules is proven.

2 Error-Robust Functional Dependencies in the RDM

We assume familarity with fundamental definitions of the RDM and functional dependencies in the RDM. One of many good sources is [9].

Suppose R is a relation schema and r, r' are R-relations. For $e \geq 0$ assume that r' results from r by introducing at most e errors per tuple. For simplicity neglect the case that r' has less elements than r, so that we can avoid considering

multisets of tuples instead of sets. We say that r satisfies the *e-error-robust functional dependency* (e-erFD) $X \rightarrow \{e\}Y$ with $X, Y \subseteq R$ iff the introduction of errors into r leading to r' would still allow to detect the functional dependency $X \rightarrow Y$. Formally, for any tuple $t' \in r'$ that corresponds to a tuple $t \in r$ there must not exist two tuples $t_1, t_2 \in r$, which both have a Hamming distance at most e from t, such that $t_1[Y] \neq t_2[Y]$ holds.

Recall that the *Hamming distance* of two tuples t_1 and t_2 (denoted as $\mathcal{H}(t_1, t_2)$) is the number of attributes B, on which $t_1[B] \neq t_2[B]$ holds.

Definition 1. Let $X, Y \subseteq R$ and $e \geq 0$. An *e-error-robust functional dependency* (e-erFD) is an expression $X \rightarrow \{e\}Y$. An R-relation r *satisfies* $X \rightarrow \{e\}Y$ iff for all R-relations r' such that there is a bijection σ between the tuples $t \in r$ and $\sigma(t) = t' \in r'$ with $\mathcal{H}(t, t') \leq e$ and all tuples $t_1, t_2 \in r$, $t' \in r'$ we have $\mathcal{H}(t_1[X], t'[X]) \leq e \wedge \mathcal{H}(t_2[X], t'[X]) \leq e \Rightarrow t_1[Y] = t_2[Y]$.

Obviously, for tuples $t_1, t_2 \in r$ with $\mathcal{H}(t_1[X], t_2[X]) \geq 2e + 1$ we obtain $t_1'[X] \neq t_2'[X]$, so these tuples cannot violate the functional dependency $X \rightarrow Y$ on r'. Conversely, for tuples $t_1, t_2 \in r$ with $\mathcal{H}(t_1[X], t_2[X]) < 2e + 1$ we may obtain $t_1'[X] = t_2'[X]$ in r', so that the tuples violate the functional dependency $X \rightarrow Y$ on r'. Using this simple fact we obtain the following easy result (see [8], also compare [4, Prop. 1.1]).

Proposition 1. *An R-relation r satisfies $X \rightarrow \{e\}Y$ iff $\mathcal{H}(t_1[X], t_2[X]) < 2e+1 \Rightarrow t_1[Y] = t_2[Y]$ holds for all tuples $t_1, t_2 \in r$.*

As in [4, p.87] we take advantage of Proposition 1 to define another class of dependencies, called *d-distance functional dependencies*, which will ease the task of finding a finite axiomatisation.

Definition 2. Let $X, Y \subseteq R$ and $d > 0$. A *d-distance functional dependency* (d-dFD) is an expression $X \rightarrow (d)Y$. An R-relation r *satisfies* $X \rightarrow (d)Y$ iff we have $\mathcal{H}(t_1[X], t_2[X]) < d \Rightarrow t_1[Y] = t_2[Y]$ for all tuples $t_1, t_2 \in r$.

As usual, we use the notation $\models_r X \rightarrow (d)Y$, if r satisfies the dFD. If Σ is a set of dFDs, we say that Σ *implies* $X \rightarrow (d)Y$ (notation: $\Sigma \models X \rightarrow (d)Y$) iff each relation r satisfying all dFDs in Σ also satisfies $X \rightarrow (d)Y$. We denote by Σ^* the *semantic hull* of Σ, i.e. the set of all dFDs implied by Σ, i.e. $\Sigma^* = \{X \rightarrow (d)Y \mid \Sigma \models X \rightarrow (d)Y\}$.

If we can find a finite, sound and complete set of rules and axioms that allows us to derive Σ^* out of Σ, then we also know how to obtain the semantic hull of a set of erFDs. This follows from the following obvious corollary of Proposition 1.

Corollary 1. *A relation r satisfies the erFD $X \rightarrow \{e\}Y$ iff r satisfies the dFD $X \rightarrow (2e + 1)Y$. In particular, 0-erFDs correspond to 1-dFDs.*

The main result on dFDs is the following theorem which was proven in a more general form in [8]. Here we use again the standard notation whereby $X, Y, Z, \ldots$ denote attribute sets, $A, B, C, \ldots$ denote attributes or attribute sets with just one attribute, and union is denoted by juxtaposition [9].

Theorem 1. *The following set $\mathfrak{R}$ of axioms and rules is sound and complete for the implication of dFDs in the RDM:*

(i) *the reflexivity axiom* $\dfrac{}{X \to (1)Y}\ Y \subseteq X$

(ii) *the weakening rule* $\dfrac{X \to (d+1)Y}{X \to (d)Y}$

(iii) *the strengthening rule* $\dfrac{X \to (d)Y}{X \to (d+1)Y}\ \mid X \mid < d$

(iv) *the union rule* $\dfrac{X \to (d)Y \quad X \to (d)Z}{X \to (d)YZ}$

(v) *the strong transitivity rule* $\dfrac{X \to (d)Y \quad YY' \to (d')Z}{X \to (d)Z}\ \mid Y' \mid < d'$

(vi) *the left strengthening rule* $\dfrac{X - A_1 \to (d)Y \ \cdots \ X - A_n \to (d)Y}{X \to (d+1)Y}\ X = \{A_1, \ldots, A_n\}$

(vii) *the left weakening rule* $\dfrac{X \to (d+1)Y}{X - A \to (d)Y}\ A \in X$

3 An Algebra of Nested Attributes

In this section we define our model of nested attributes, which covers the gist of higher-order datamodels including HERM, the OODM and XML. In particular, we investigate the structure of the set $S(X)$ of subattributes of a given nested attribute X, which will give us a Brouwer algebra [6, 7].

3.1 Nested Attributes

We start with a definition of simple attributes and values for them.

Definition 3. A *universe* is a finite set $\mathcal{U}$ together with domains (i.e. sets of values) $dom(A)$ for all $A \in \mathcal{U}$. The elements of $\mathcal{U}$ are called *simple attributes.*

For the relational model a universe was enough, as a relation schema could be defined by a subset $R \subseteq \mathcal{U}$. For higher-order datamodels, however, we need nested attributes. In the following definition we use a set $\mathcal{L}$ of labels, and tacitly assume that the symbol λ is neither a simple attribute nor a label, i.e. $\lambda \notin \mathcal{U} \cup \mathcal{L}$, and that simple attributes and labels are pairwise different, i.e. $\mathcal{U} \cap \mathcal{L} = \emptyset$.

Definition 4. Let $\mathcal{U}$ be a universe and $\mathcal{L}$ a set of labels. The set $\mathcal{N}$ of *nested attributes* (over $\mathcal{U}$ and $\mathcal{L}$) is the smallest set with $\lambda \in \mathcal{N}$, $\mathcal{U} \subseteq \mathcal{N}$, and satisfying the following properties:

- for $X \in \mathcal{L}$ and $X'_1, \ldots, X'_n \in \mathcal{N}$ we have $X(X'_1, \ldots, X'_n) \in \mathcal{N}$;
- for $X \in \mathcal{L}$ and $X' \in \mathcal{N}$ we have $X\{X'\} \in \mathcal{N}$, $X[X'] \in \mathcal{N}$, and $X\langle X'\rangle \in \mathcal{N}$.

We call λ a *null attribute*, $X(X'_1, \ldots, X'_n)$ a *record attribute*, $X\{X'\}$ a *set attribute*, $X[X']$ a *list attribute*, and $X\langle X'\rangle$ a *multiset attribute*. As record, set, list and multiset attributes have a unique leading label, say X, we often write simply X to denote the attribute.

We can now extend the association *dom* from simple to nested attributes, i.e. for each $X \in \mathcal{N}$ we will define a set of values $dom(X)$.

Definition 5. For each nested attribute $X \in \mathcal{N}$ we get a *domain* $dom(X)$ as follows:

- $dom(\lambda) = \{\top\}$;
- $dom(X(X'_1, \ldots, X'_n)) = \{(X_1 : v_1, \ldots, X_n : v_n) \mid v_i \in dom(X'_i) \text{ for } i = 1, \ldots, n\}$ with labels X_i for the attributes X'_i;
- $dom(X\{X'\}) = \{\{v_1, \ldots, v_n\} \mid v_i \in dom(X') \text{ for } i = 1, \ldots, n\}$, i.e. each element in $dom(X\{X'\})$ is a finite set with elements in $dom(X')$;
- $dom(X[X']) = \{[v_1, \ldots, v_n] \mid v_i \in dom(X') \text{ for } i = 1, \ldots, n\}$, i.e. each element in $dom(X[X'])$ is a finite list with elements in $dom(X')$;
- $dom(X\langle X'\rangle) = \{\langle v_1, \ldots, v_n\rangle \mid v_i \in dom(X') \text{ for } i = 1, \ldots, n\}$, i.e. each element in $dom(X\langle X'\rangle)$ is a finite multiset with elements in $dom(X')$.

Note that the relational model is covered, if only the tuple constructor is used. Thus, instead of a relation schema R we will now consider a nested attribute X, assuming that the universe $\mathcal{U}$ and the set of labels $\mathcal{L}$ are fixed. Instead of an R-relation r we will consider a finite set $r \subseteq dom(X)$.

3.2 Subattributes

In the dependency theory for the relational model we exploited projections on subsets X of a relation schema R. These are just special cases of projections on subattributes. Therefore, we will define a partial order $\geq$ on $\mathcal{N}$. However, this partial order will be defined on equivalence classes of attributes. We will identify nested attributes, if we can identify their domains.

Definition 6. $\equiv$ is the smallest *equivalence relation* on $\mathcal{N}$ satisfying the following properties:

- $\lambda \equiv X()$;
- $X(X'_1, \ldots, X'_n) \equiv X(X'_1, \ldots, X'_n, \lambda)$;
- $X(X'_1, \ldots, X'_n) \equiv X(X'_{\sigma(1)}, \ldots, X'_{\sigma(n)})$ for any permutation σ;

- $X(X_1', \ldots, X_n') \equiv X(Y_1, \ldots, Y_n)$ iff $X_i' \equiv Y_i$ for all $i = 1, \ldots, n$;
- $X\{X'\} \equiv X\{Y\}$ iff $X' \equiv Y$;
- $X[X'] \equiv X[Y]$ iff $X' \equiv Y$;
- $X\langle X' \rangle \equiv X\langle Y \rangle$ iff $X' \equiv Y$.

Basically, the equivalence definition states that λ in record attributes can be added or removed, and that order in record and union attributes does not matter.

In the following we identify $\mathcal{N}$ with the set $\mathcal{N}/_{\equiv}$ of equivalence classes. In particular, we will write $=$ instead of $\equiv$, and in the following definition we should say that Y is a subattribute of X iff $\tilde{X} \geq \tilde{Y}$ holds for some $\tilde{X} \equiv X$ and $\tilde{Y} \equiv Y$.

Definition 7. For $X, Y \in \mathcal{N}$ we say that Y is a *subattribute* of X, iff $X \geq Y$ holds, where $\geq$ is the smallest partial order on $\mathcal{N}$ satisfying the following properties:

- $X \geq \lambda$ for all $X \in \mathcal{N}$;
- $X(Y_1, \ldots, Y_n) \geq X(X'_{\sigma(1)}, \ldots, X'_{\sigma(m)})$ for some injective $\sigma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $Y_{\sigma(i)} \geq X'_{\sigma(i)}$ for all $i = 1, \ldots, m$;
- $X\{Y\} \geq X\{X'\}$ iff $Y \geq X'$;
- $X[Y] \geq X[X']$ iff $Y \geq X'$;
- $X\langle Y \rangle \geq X\langle X' \rangle$ iff $Y \geq X'$.

Obviously, $X \geq Y$ induces a projection map $\pi_Y^X : dom(X) \to dom(Y)$. For $X \equiv Y$ we have $X \geq Y$ and $Y \geq X$ and the projection maps π_Y^X and π_X^Y are inverse to each other.

We use the notation $\mathcal{S}(X) = \{Z \in \mathcal{N} \mid X \geq Z\}$ to denote the *set of subattributes* of a nested attribute X. It has been shown that $\mathcal{S}(X)$ carries the structure of a Brouwer algebra [5, 6, 7].

Proposition 2. *The set $\mathcal{S}(X)$ of subattributes carries the structure of a Brouwer algebra, i.e. it is a distributive lattice with a meet-operation $\sqcap$, a join-operation $\sqcup$, a smallest element λ, a largest element X, and relative pseudo-complements $Y \leftarrow Z = \bigsqcap\{U \mid U \sqcup Y \geq Z\}$.*

Figure 1 as an example shows the Brouwer algebra $\mathcal{S}(X(X_1\{A\}, X_2[B]))$.

3.3 Ideals of Subattributes

We are dealing with several constructors for complex values at the same time. In order to cope with the problems arising from this fact, we need some additional notions that we will define in this subsection.

For the derivation rules for functional dependencies we need a notion of when two subattributes are "nearly disjoint". This property is called *semi-disjointness*.

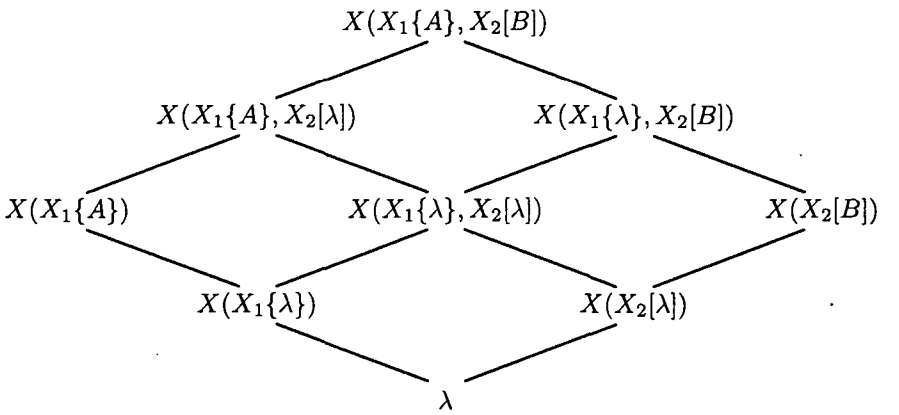

Figure 1: The lattice $\mathcal{S}(X(X_1\{A\}, X_2[B]))$

Definition 8. Two subattributes $Y, Z \in \mathcal{S}(X)$ are called *semi-disjoint* iff one of the following holds:

(i) $Y \geq Z$ or $Z \geq Y$;

(ii) $X = X(X_1, \ldots, X_n)$, $Y = X(Y_1, \ldots, Y_n)$, $Z = X(Z_1, \ldots, Z_n)$ and $Y_i, Z_i \in \mathcal{S}(X_i)$ are semi-disjoint for all $i = 1, \ldots, n$;

(iii) $X = X[X']$, $Y = X[Y']$, $Z = X[Z']$ and $Y', Z' \in \mathcal{S}(X')$ are semi-disjoint.

For the soundness proof in the next section we will need the following simple fact about projections to semi-disjoint attributes.

Lemma 1. *Let $t_1, t_2 \in dom(X)$ for some nested attribute $X \in \mathcal{N}$ such that $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ and $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ hold for semi-disjoint subattributes $Y, Z \in \mathcal{S}(X)$. Then also $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$ holds.*

Proof. We use induction on X to show $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$. The cases $X = \lambda$ and $X = A$ (i.e. a simple attribute) are trivial. There is also nothing to show for $Y \geq Z$ or $Z \geq Y$, as in these cases $Y \sqcup Z$ is one of Y or Z.

For $X = X(X_1, \ldots, X_n)$, semi-disjoint $Y = X(Y_1, \ldots, Y_n)$ and $Z = X(Z_1, \ldots, Z_n)$, and $t_j = (X_1 : t_{j1}, \ldots, X_n : t_{jn})$ $(j = 1, 2)$ we have $\pi^{X_i}_{Y_i}(t_{1i}) = \pi^{X_i}_{Y_i}(t_{2i})$ and $\pi^{X_i}_{Z_i}(t_{1i}) = \pi^{X_i}_{Z_i}(t_{2i})$, and Y_i, Z_i are semi-disjoint for all $i = 1, \ldots, n$. By induction $\pi^{X_i}_{Y_i \sqcup Z_i}(t_{1i}) = \pi^{X_i}_{Y_i \sqcup Z_i}(t_{2i})$, which implies $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$.

For $X = X[X']$ and semi-disjoint $Y = X[Y']$ and $Z = X[Z']$, the subattributes Y' and Z' of X' are also semi-disjoint. Furthermore, for $t_j = [t_{j1}, \ldots, t_{jn_j}]$ $(j = 1, 2)$ we must have $n_1 = n_2$ and $\pi^{X'}_{Y'}(t_{1k}) = \pi^{X'}_{Y'}(t_{2k})$ and $\pi^{X'}_{Z'}(t_{1k}) = \pi^{X'}_{Z'}(t_{2k})$ for all $k = 1, \ldots, n_1$. By induction we get also $\pi^{X'}_{Y' \sqcup Z'}(t_{1k}) = \pi^{X'}_{Y' \sqcup Z'}(t_{2k})$ for all $k = 1, \ldots, n_1$, i.e. $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$. □

As $\mathcal{S}(X)$ is a lattice, it makes sense to investigate ideals and filters. The following notion of an *HL-ideal* will be central for the completeness proof in the next section.

Definition 9. Let $X \in \mathcal{N}$. An *HL-ideal* on $\mathcal{S}(X)$ is a subset $\mathcal{F} \subseteq \mathcal{S}(X)$ with the following properties:

(i) $\lambda \in \mathcal{F}$;

(ii) if $Y \in \mathcal{F}$ and $Z \in \mathcal{S}(X)$ with $Y \geq Z$, then $Z \in \mathcal{F}$;

(iii) if $Y, Z \in \mathcal{F}$ are semi-disjoint, then $Y \sqcup Z \in \mathcal{F}$.

The key step in the completeness proof for dFDs in the RDM in [8] consists in the construction of a relation with exactly two tuples, which coincide exactly on a given set of attributes. While this is trivial for the RDM, the presence of complex values requires a similar result. However, instead of a set of attributes we now have to deal with an HL-ideal. This result — denoted *Central Lemma* in [6] — provides the major difficulty for the axiomatisation of functional dependencies in [5], [7] and [6].

The following theorem states this result for the case that we deal with records, lists, sets and multisets. The non-trivial lengthy proof is contained in [7].

Theorem 2. *Let $X \in \mathcal{N}$ and $\mathcal{F}$ be an* HL-ideal *on $\mathcal{S}(X)$. Then there exist $t_1, t_2 \in dom(X)$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ iff $Y \in \mathcal{F}$.*

4 Distance Functional Dependencies on Nested Attributes

Our major goal is to generalise Theorem 1 to dFDs on nested attributes. Therefore, we first have to generalise FDs and dFDs to this case. For the latter ones the major difficulty is to define a generalisation of the Hamming distance for complex values.

4.1 A Generalised Distance Function on Complex Values

Let us first define ordinary functional dependencies. As a set of attributes in the RDM corresponds to a single record attribute, the first idea is to replace sets of attributes by a single subattribute. While this is sufficient for records and lists, it is not a good idea for sets and multisets. The reason is that the well known extension rule in Armstrong's axiomatisation for FDs in the RDM does not generalise in this way [5]. Therefore, we have to consider sets of subattributes instead.

Definition 10. Let $X \in \mathcal{N}$. A *functional dependency* (FD) on $\mathcal{S}(X)$ is an expression $\mathcal{Y} \to \mathcal{Z}$ with $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X)$.

Let r be an instance of X. We say that r *satisfies the FD* $\mathcal{Y} \to \mathcal{Z}$ on $\mathcal{S}(X)$ (notation: $r \models \mathcal{Y} \to \mathcal{Z}$) iff for all $t_1, t_2 \in r$ with $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ for all $Y \in \mathcal{Y}$ we also have $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$.

Now recall, that the difference between a FD and a dFD in the RDM was that we replaced the equality on the left hand side by a bound on the Hamming distance bewtwen two tuples with the distance $d = 1$ corresponding to the case of an ordinary FD. The Hamming distance counts the number of attributes, on which two tuples differ. These attributes form some kind of a "basis" in the sense that each subset of a relation schema can be constructed as a union of singleton sets containing just one attribute.

In order to generalise the distance notion to complex values, we therefore need a basis made of subattributes.

Definition 11. Let $X \in \mathcal{N}$. The *subattribute basis of* X is the smallest subset $\mathcal{B}(X) \subseteq \mathcal{S}(X)$ such that each $Y \in \mathcal{S}(X)$ can be written in the form $Y = \bigsqcup_{Y' \in \mathcal{B}_Y} Y'$ for some $\mathcal{B}_Y \subseteq \mathcal{B}(X)$.

The subattribute basis of a simple record attribute would just give us the simple attributes. Therefore, considering the subattribute basis suggests to be a good choice to replace the set of attributes in the definition of the distance function. However, in order to cope properly with sets ans multisets, we need to close $\mathcal{B}(X)$ under the join of attributes that are not semi-disjoint.

Definition 12. Let $X \in \mathcal{N}$. The *Hamming basis of* X is the smallest subset $\mathcal{C}(X) \subseteq \mathcal{S}(X)$ with $\mathcal{B}(X) \subseteq \mathcal{C}(X)$ such that for all non-semi-disjoint $Y, Z \in \mathcal{C}(X)$ we also have $Y \sqcup Z \in \mathcal{C}(X)$.

The following is an easy implication of Lemma 1.

Lemma 2. *Let $t_1, t_2 \in dom(X)$ for some nested attribute X and $Y \in \mathcal{X}$. If $\pi^X_{Y'}(t_1) = \pi^X_{Y'}(t_2)$ holds for all $Y' \in \mathcal{C}(Y)$, then also $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ holds.*

Proof. According to the definition of the subattribute basis $\mathcal{B}(X)$ and the Hamming basis $\mathcal{C}(X)$ we can write Y in the form $Y = \bigsqcup_{Y' \in \mathcal{C}_Y} Y'$ for some subset $\mathcal{C}_Y \subseteq \mathcal{C}(X)$. By definition the elements in $\mathcal{C}_Y$ are pairwise semi-disjoint. As t_1 and t_2 coincide on all elements of $\mathcal{C}_Y$, the also coincide on Y by Lemma 1. □

Now we can use the Hamming basis of X to define the distance of two complex values $t_1, t_2 \in dom(X)$.

Definition 13. Let $X \in \mathcal{N}$ and $t_1, t_2 \in dom(X)$. The *Hamming distance* $\mathcal{H}(t_1, t_2)$ between t_1 and t_2 is defined as $\mathcal{H}(t_1, t_2) = |\{Y \in \mathcal{C}(X) \mid \pi^X_Y(t_1) \neq \pi^X_Y(t_2)\}|$, i.e. as the number of subattributes in the Hamming basis, on which t_1 and t_2 differ.

This leads us straightforward to the generalisation of dFDs on a nested attribute.

Definition 14. Let $X \in \mathcal{N}$ be a nested attribute and $d \geq 1$. A *d-distance functional dependency* (dFD) on $\mathcal{S}(X)$ is an expression of the form $\mathcal{Y} \rightarrow (d)\mathcal{Z}$ with $\mathcal{Y}, \mathcal{Z} \subseteq \mathcal{S}(X)$.

Let r be an instance of X. We say that r *satisfies the dFD* $\mathcal{Y} \rightarrow (d)\mathcal{Z}$ on $\mathcal{S}(X)$ (notation: $r \models \mathcal{Y} \rightarrow (d)\mathcal{Z}$) iff for all $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < d$ for all $Y \in \mathcal{Y}$ we also have $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$.

As before, we use $\models$ to denote implication of dFDs and Σ^* to denote the semantic hull of a set Σ of dFDs.

4.2 Sound Derivation Rules

Using the definitions from the last subsection we will show now that derivation rules similar to the ones in Theorem 1 are sound for the implication of dFDs on nested attributes. Before we can define this set of derivation rules, we need a few more notation.

For $Y \in \mathcal{Y} \subseteq \mathcal{S}(X)$ let $\downarrow Y = \{Y' \mid Y' \text{ maximal with } Y \geq Y'\}$. Furthermore, if $\mathcal{Y} = \{Y_1, \ldots, Y_k\}$, write $\downarrow \mathcal{Y} = \{\{Y_1', \ldots, Y_k'\} \mid Y_i' \in \downarrow Y_i \text{ for one } i \text{ and } Y_j' = Y_j \text{ for all } j \neq i\}$. In particular we have a mapping $Y_i \mapsto Y_i'$ and we can define $k_i = |\mathcal{C}(Y_i)| - |\mathcal{C}(Y_i')|$. We use this to define $k(\mathcal{Y}, \mathcal{Y}') = \max k_i$ for $\mathcal{Y}' = \{Y_1', \ldots, Y_k'\} \in \downarrow \mathcal{Y}$.

Theorem 3. *Let $X \in \mathcal{N}$ be a nested attribute. The following rules are sound for the implication of dFDs on $\mathcal{S}(X)$:*

reflexivity axiom:
$$\frac{}{\mathcal{Y} \to (1)\mathcal{Z}}\ \mathcal{Z} \subseteq \mathcal{Y} \tag{1}$$

lambda axiom:
$$\frac{}{\emptyset \to (d)\{\lambda\}} \tag{2}$$

subattribute axiom:
$$\frac{}{\{Y\} \to (1)\{Z\}}\ Y \geq Z \tag{3}$$

join axiom:
$$\frac{}{\{Y, Z\} \to (1)\{Y \sqcup Z\}}\ Y, Z \text{ semi-disjoint} \tag{4}$$

weakening rule:
$$\frac{\mathcal{Y} \to (d+1)\mathcal{Z}}{\mathcal{Y} \to (d)\mathcal{Z}} \tag{5}$$

strengthening rule:
$$\frac{\mathcal{Y} \to (d)\mathcal{Z}}{\mathcal{Y} \to (d+1)\mathcal{Z}}\ \max\{|\mathcal{C}(Y)| \mid Y \in \mathcal{Y}\} < d \tag{6}$$

union rule:
$$\frac{\mathcal{Y} \to (d)\mathcal{Z}_1 \quad \mathcal{Y} \to (d)\mathcal{Z}_2}{\mathcal{Y} \to (d)\mathcal{Z}_1 \cup \mathcal{Z}_2} \tag{7}$$

strong transitivity rule:
$$\frac{\mathcal{Y} \to (d)\mathcal{Z} \quad \mathcal{Z} \cup \mathcal{Z}' \to (d')\mathcal{U}}{\mathcal{Y} \to (d)\mathcal{U}}\ \max\{|\mathcal{C}(Z)| \mid Y \in \mathcal{Z}'\} < d' \tag{8}$$

left strengthening rule:
$$\frac{\mathcal{Y}_1 \to (d-k_1)\mathcal{Z} \ \ldots\ \mathcal{Y}_m \to (d-k_m)\mathcal{Z}}{\mathcal{Y} \to (d)\mathcal{Z}} \tag{9}$$
for $\downarrow \mathcal{Y} = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_m\}$ and $k_i = k(\mathcal{Y}, \mathcal{Y}_i)$

left weakening rule:
$$\frac{\mathcal{Y} \to (d+k)\mathcal{Z}}{\mathcal{Y}' \to (d)\mathcal{Z}}\ \mathcal{Y}' \in \downarrow \mathcal{Y}, k = k(\mathcal{Y}, \mathcal{Y}') \tag{10}$$

Proof. In the following let r be an instance of X, i.e. $r \subseteq dom(X)$. The soundness of the weakening rule (5) is obvious.

For the soundness of the reflexivity axiom (1) let $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < 1$ for all $Y \in \mathcal{Y}$. That is, t_1 and t_2 coincide on all subattributes in $\mathcal{C}(Y)$ for all $Y \in \mathcal{Y}$. As $\mathcal{Z} \subseteq \mathcal{Y}$ holds, they must also coincide on all subattributes in $\mathcal{C}(Z)$ for all $Z \in \mathcal{Z}$.

The soundness of the lambda-axiom (2) is obvious, as any $t_1, t_2 \in r$ coincide on λ.

If t_1 and t_2 coincide on all subattributes in $\mathcal{C}(Y)$, they also coincide on Y by Lemma 2, and as $Y \geq Z$, they must also coincide on Z, which proves the soundness of the subattribute-axiom (3).

Similarly, $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < 1$ and $\mathcal{H}(\pi^X_Z(t_1), \pi^X_Z(t_2)) < 1$ implies that t_1 and t_2 coincide on all subattributes in $\mathcal{C}(Y) \cup \mathcal{C}(Z)$. By Lemma 2 they must also coincide on Y and Z. As Y, Z are semi-disjoint, we obtain $\pi^X_{Y \sqcup Z}(t_1) = \pi^X_{Y \sqcup Z}(t_2)$ by Lemma 1, which proves the soundness of the join-axiom (4).

For the soundness of the strengthening rule (6) take $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < d + 1$ for all $Y \in \mathcal{Y}$. As $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < |\mathcal{C}(Y)| < d$ for all $Y \in \mathcal{Y}$, the premise implies $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$ as claimed.

For the soundness of the union rule (7) take $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < d$ for all $Y \in \mathcal{Y}$. The premises of the rule imply $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}_j$ $(j = 1, 2)$, which trivially implies $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}_1 \cup \mathcal{Z}_2$.

In order to prove the soundness of the strong transitivity rule (8) take again $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < d$ for all $Y \in \mathcal{Y}$. The first premise of the rule implies $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$. For $Z' \in \mathcal{Z}'$ we have $\mathcal{H}(\pi^X_{Z'}(t_1), \pi^X_{Z'}(t_2)) < |\mathcal{C}(\mathcal{Z}')| < d'$. Hence $\mathcal{H}(\pi^X_Z(t_1), \pi^X_Z(t_2)) < d'$ for all $Z \in \mathcal{Z} \cup \mathcal{Z}'$. The second premise of the rule gives the desired $\pi^X_U(t_1) = \pi^X_U(t_2)$ for all $U \in \mathcal{U}$.

Now take again $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < d$ for all $Y \in \mathcal{Y}$. Unless $\pi^X_Y(t_1) = \pi^X_Y(t_2)$ there must exist some $\mathcal{Y}_i \in \downarrow \mathcal{Y}$ with $\mathcal{H}(\pi^X_{Y'}(t_1), \pi^X_{Y'}(t_2)) < d - k_i$ for all $Y' \in \mathcal{Y}_i$, and we can apply the corresponding premise of the left strengthening rule (9) to conclude $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$, which proves the soundness of this rule.

Finally, for the soundness of the left weakening rule (10) take again $t_1, t_2 \in r$ with $\mathcal{H}(\pi^X_{Y'}(t_1), \pi^X_{Y'}(t_2)) < d$ for all $Y' \in \mathcal{Y}'$. Hence, $\mathcal{H}(\pi^X_Y(t_1), \pi^X_Y(t_2)) < d + 1$ for all $Y \in \mathcal{Y}$. Applying the premise of the rule leads to $\pi^X_Z(t_1) = \pi^X_Z(t_2)$ for all $Z \in \mathcal{Z}$ as claimed. □

4.3 Completeness

As usual, given a set of axioms and rules $\mathfrak{R}$, and a set Σ of dFDs, we let Σ^+ denote the *syntactic hull* of Σ, i.e. the set of all dFDs that can be derived from Σ using the axioms and rules in $\mathfrak{R}$. In the following we take $\mathfrak{R}$ as the axioms and rules from Theorem 3. This theorem already states the soundness of $\mathfrak{R}$, i.e. $\Sigma^+ \subseteq \Sigma^*$.

A set of axioms and rules is called *complete* iff $\Sigma^* \subseteq \Sigma^+$ holds. Our final goal is to show the completeness of the rules in $\mathfrak{R}$. Theorem 2 will turn out to be the key for the completeness proof in this section.

Theorem 4. *The set $\mathfrak{R}$ of axioms and rules from Theorem 3 is complete for the implication of dFDs on nested attributes.*

Proof. Let $X \in \mathcal{N}$ be a nested attribute, and let Σ denote a set of dFDs on $\mathcal{S}(X)$. In order to show $\Sigma^* \subseteq \Sigma^+$ let $\mathcal{Y} \to (d)\mathcal{Z} \notin \Sigma^+$.

Let d be minimal with this property. Then, according to rule (6) we can assume that $|\mathcal{C}(Y)| \geq d-1$ for at least one $Y \in \mathcal{Y}$. Otherwise, we would have $\max\{|\mathcal{C}(Y)| \mid Y \in \mathcal{Y}\} < d-1$. As d is minimal, we have $\mathcal{Y} \to (d-1)\mathcal{Z} \in \Sigma^+$, and applying the strengthening rule (6) would result in the contradiction $\mathcal{Y} \to (d)\mathcal{Z} \in \Sigma^+$.

Due to the union rule (7) there must be some $Z \in \mathcal{Z}$ with $\mathcal{Y} \to (d)\{Z\} \notin \Sigma^+$. Then, applying the left strengthening rule (9) k times with $k \leq d-1$ — which is possible, as $|\mathcal{C}(Y)| \geq d-1$ for at least one $Y \in \mathcal{Y}$ — we find some $\mathcal{Y}' \in \downarrow^k \mathcal{Y}$ with $\mathcal{Y}' \to (1)\{Z\} \notin \Sigma^+$.

Now take $\mathcal{Y}'^+ = \{U \mid \mathcal{Y}' \to (1)\{U\} \in \Sigma^+\}$, so $Z \notin \mathcal{Y}'^+$, but due to the reflexivity axiom (1) we have $\mathcal{Y}' \subseteq \mathcal{Y}'^+$.

Obviously, due to the lambda axiom (2), the subattribute axiom (3) and the join axiom (4) $\mathcal{Y}'^+$ is an HL-ideal in the Brouwer algebra $\mathcal{S}(X)$. Applying Theorem 2 to $\mathcal{Y}'^+$ we obtain an instance $r = \{t_1, t_2\}$ such that $\pi_U^X(t_1) = \pi_U^X(t_2)$ holds iff $U \in \mathcal{Y}'^+$.

Hence, $r \not\models \mathcal{Y}' \to (1)\{Z\}$, and applying the sound left weakening rule (10) k times we obtain $r \not\models \mathcal{Y} \to (d)\{Z\}$. From the soundness of the reflexivity axiom (1) we further obtain $r \not\models \mathcal{Y} \to (d)\mathcal{Z}$.

We now show $r \models \Sigma$. So let $\mathcal{U} \to (d')\mathcal{V} \in \Sigma$. We consider two cases:

(i) Assume $\max\{|\mathcal{C}(U)| \mid U \in \mathcal{U}\} < d'$. Due to the lambda rule (2) we have $\mathcal{Y}' \to (1)\{\lambda\} \in \Sigma^+$. Using the strong transitivity rule (8) with $\mathcal{Z}' = \mathcal{U}$, we obtain $\mathcal{Y}' \to (1)\mathcal{V} \in \Sigma^+$, hence $\mathcal{V} \subseteq \mathcal{Y}'^+$. Due to the construction of r we obtain $\pi_V^X(t_1) = \pi_V^X(t_2)$ for all $V \in \mathcal{V}$, which shows $r \models \mathcal{U} \to (d')\mathcal{V}$.

(ii) Next assume $\max\{|\mathcal{C}(U)| \mid U \in \mathcal{U}\} \geq d'$. We show $r \models \mathcal{U}' \to (1)\mathcal{V}$, whenever $\mathcal{U}' \in \downarrow^{k'} \mathcal{U}$ with $k' \leq d'-1$, and $\mathcal{U}' \to (1)\mathcal{V} \in \Sigma^+$ results from applying the left weakening rule (10) k' times.

Then the soundness of the left strengthening rule (9) implies $r \models \mathcal{U} \to (d')\mathcal{V}$ as desired.

We distinguish again two subcases:

(a) If $\mathcal{U}' \not\subseteq \mathcal{Y}'^+$, we have $\pi_{U'}^X(t_1) \neq \pi_{U'}^X(t_2)$ for at least one $U' \in \mathcal{U}'$, which immediately implies $r \models \mathcal{U}' \to (1)\mathcal{V}$.

(b) If $\mathcal{U}' \subseteq \mathcal{Y}'^+$, we have $\mathcal{Y}' \to (1)\{U'\} \in \Sigma^+$ for all $U' \in \mathcal{U}'$. Using the union rule (7) we conclude $\mathcal{Y}' \to (1)\mathcal{U}' \in \Sigma^+$, and further $\mathcal{Y}' \to (1)\mathcal{V} \in \Sigma^+$ by applying the strong transitivity rule (8).
Hence $\mathcal{V} \subseteq \mathcal{Y}'^+$, which implies $r \models \mathcal{U}' \to (1)\mathcal{V}$ as desired.

Now $r \models \Sigma^*$, but $r \not\models \mathcal{Y} \to (d)\mathcal{Z}$. Hence $\mathcal{Y} \to (d)\mathcal{Z} \notin \Sigma^*$, which completes the proof. □

5 Conclusion

In this article we presented a finite axiomatisation of distance functional dependencies on nested attributes. This result generalises a corresponding result for the RDM that was achieved (in a more general form) in [8].

The major tasks to solve this problem were generalising the Hamming distance from tuples to arbitrary complex values, and constructing values that coincide exactly on a given ideal of subattributes. The latter problem was solved in [7] with solutions to a subcase contained in [5].

For the generalisation of the Hamming distance we used the "Hamming basis", which results from the subattribute basis by adding the joins of all non-semi-disjoint subattributes. This preserves the Hamming distance on flat tuples as a special case. That is, the new Hamming distance counts the number of subattributes in the Hamming basis, on which two values differ.

Alternatively, we could have chosen all subattributes instead of just those in the Hamming basis. Looking through the proofs in this article, this would not have affected the finite axiomatisation. However, we would have obtained a distance function with significant jumps.

We might still feel that the new distance function is still too coarse, as it cannot express counting. For instance, two sets with elements in the domain of a simple attribute either have distance 0, i.e. they are equal, or 1, i.e. they are different but both non-empty, or 2, i.e. they are different and one of thre sets is empty. However, the same problem appears already with functional dependencies, and thus, has to be solved in a larger context.

References

[1] S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers 2000.

[2] W.W. Armstrong. Dependency Structures of Database Relationships. *Information Processing* vol. 74: 580-583, 1974.

[3] J. Demetrovics, G.O.H. Katona, D. Miklós. Error-Correcting Keys in Relational Databases. in K.-D. Schewe, B. Thalheim (Eds.). *Foundations of Information and Knowledge Systems. First International Symposium, FoIKS 2000.* Springer-Verlag, LNCS vol. 1762: 88-93. Berlin 2000.

[4] J. Demetrovics, G.O.H. Katona, D. Miklós. Functional Dependencies in Presence of Errors. in T. Eiter, K.-D. Schewe (Eds.). *Foundations of Information and Knowledge Systems. Second International Symposium, FoIKS 2002.* Springer-Verlag, LNCS vol. 2284: 85-92. Berlin 2002.

[5] S. Hartmann, A. Hoffmann, S. Link, K.-D. Schewe. Axiomatizing Functional Dependencies in the Higher-Order Entity-Relationship Model. *Information Processing Letters* vol. 87 (2003): 133-137.

[6] S. Hartmann, S. Link, K.-D. Schewe. Weak Functional Dependencies in Higher-Order Datamodels – The Case of the Union Constructor. in D. Seipel, J. M. Turull Torres (Eds.). *Foundations of Information and Knowledge Systems. Third International Symposium, FoIKS 2004*. Springer-Verlag LNCS vol. 2942: 117-134. Berlin 2004.

[7] S. Hartmann, S. Link, K.-D. Schewe. Axiomatisation of Functional Dependencies in the Presence of Records, Lists, Sets and Multisets. Massey University 2003. submitted for publication.

[8] S. Hartmann, S. Link, K.-D. Schewe, B.Thalheim. Error-Robust Functional Dependencies. Massey University 2002. submitted for publication.

[9] J. Paredaens, P. De Bra, M. Gyssens, D. Van Gucht. *The Structure of the Relational Database Model*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin Heidelberg 1989.

[10] K.-D. Schewe and B. Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica* vol. 11 (4): 49-85, 1993.

[11] B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag, Berlin Heidelberg 2000.

Received October, 2002

Acta Cybernetica **16** (2004) 473–483.

Some Problems Related to Keys and the Boyce-Codd Normal Form

Vu Duc Thi* and Nguyen Hoang Son†

Abstract

The aim of this paper is to investigate the connections between minimal keys and antikeys for special Sperner-systems by hypergraphs. The Boyce-Codd normal form and some related problems are also studied in this paper.

1 Introduction

In the relational datamodel, one of the important concepts is the functional dependency. Several types of families of functional dependencies which satisfy some conditions are known under the name of normal forms (NFs). The most desirable NF is Boyce-Codd NF (BCNF) that has been investigated in a lot of papers (see [2, 8, 9, 10]). The minimal keys and set of antikeys are interesting concepts in the relational datamodel (see, e.g., [11, 12]). A set of minimal keys and set of antikeys form Sperner-systems. Sperner-systems and sets of minimal keys are equivalent in the sense that for an arbitrary Sperner-system K a family of functional dependencies F can be constructed so that the minimal keys of F are exactly the elements of K (see [5]).

Hypergraph theory (see, e.g., [3]) is an important subfield of discrete mathematics with many relevant applications in both theoretical and applied computer science. The transversal and the minimal transversal of a hypergraph are important concepts in this theory, on one hand.

The paper is structured as follows: in the second section, some necessary definitions and results about hypergraph theory are given.

In Section 3, transformations of the notions and the results of Section 2 concerning hypergraphs to relational databases are shown. We prove that the set of all prime attributes is the set of all independent attributes of a given relation scheme. We give an effective algorithm finding a BCNF relation r such that r represents a given BCNF relation scheme s (i.e., $K_r = K_s$, where K_r and K_s are sets of all minimal keys of r and s). We aslo give an effective algorithm which from a given BCNF

*Institute of Information Technology, National Centre for Natural Science and Technology of Vietnam, 18 Hoang Quoc Viet, Hanoi, Vietnam

†Department of Mathematics, College of Sciences, Hue University, Vietnam

relation r finds a BCNF relation scheme s such that $K_r = K_s$. Section 4, we study the connections between minimal keys and antikeys for special Sperner-system by hypergraphs.

2 Basic definitions and results

In this section we start with some basic definitions and results on hypergraphs.

Definition 2.1. *Let R be a nonempty finite set and put $\mathcal{P}(R)$ for the family of all subsets of R (its power set). The family $\mathcal{H} = \{E_i : E_i \in \mathcal{P}(R), i = 1, ..., m\}$ is called a hypergraph over R if $E_i \neq \emptyset$ holds for all i (in [3] it is required that the union of E_is is R, in this paper we do not require this).*

The elements of R are called vertices, and the sets $E_1, ..., E_m$ the edges of the hypergraph $\mathcal{H}$.

A hypergraph $\mathcal{H}$ is called simple if it satisfies $\forall E_i, E_j \in \mathcal{H} : E_i \subseteq E_j \Rightarrow E_i = E_j$. It can be seen that simple hypergraphs are Sperner-systems.

One can see easily that the family $m(\mathcal{H}) = \{E_i \in \mathcal{H} : \nexists E_j \in \mathcal{H} : E_j \subset E_i\}$ is a simple hypergraph, and that $m(\mathcal{H})$ is uniquely determined by $\mathcal{H}$.

Definition 2.2. *Let $\mathcal{H}$ be a hypergraph over R. A set $T \subseteq R$ is called a transversal of $\mathcal{H}$ (sometimes it is called hitting set) if it meets all edges of $\mathcal{H}$, i.e., $\forall E \in \mathcal{H} : T \cap E \neq \emptyset$. Denote by $Trs(\mathcal{H})$ the family of all transversals of $\mathcal{H}$. A transversal T of $\mathcal{H}$ is called minimal if no proper subset T' of T is a transversal.*

The family of all minimal transversals of $\mathcal{H}$ called the transversal hypergraph of $\mathcal{H}$, and denoted by $Tr(\mathcal{H})$. Clearly, $Tr(\mathcal{H})$ is a simple hypergraph.

The following algorithm finds the family of all minimal transversals of a given hypergraph (by induction).

Algorithm 2.1. (Demetrovics and Thi [7]).
Input: Let $\mathcal{H} = \{E_1, ..., E_m\}$ be a hypergraph over R.
Output: $Tr(\mathcal{H})$.
Method:
Step 0: We set $L_1 := \{\{a\} : a \in E_1\}$. It is obvious that $L_1 = Tr(\{E_1\})$.
Step q+1: $(q < m)$ Assume that

$$L_q = S_q \cup \{B_1, ..., B_{t_q}\},$$

where $B_i \cap E_{q+1} = \emptyset, i = 1, ..., t_q$ and $S_q = \{A \in L_q : A \cap E_{q+1} \neq \emptyset\}$.

For each i $(i = 1, ..., t_q)$ constructs the set $\{B_i \cup \{b\} : b \in E_{q+1}\}$. Denote them by $A_1^i, ..., A_{r_i}^i (i = 1, ..., t_q)$. Let

$$L_{q+1} = S_q \cup \{A_p^i : A \in S_q \Rightarrow A \not\subset A_p^i, 1 \leq i \leq t_q, 1 \leq p \leq r_i\}.$$

Theorem 2.1. (Demetrovics and Thi [7]). *For every q $(1 \leq q \leq m)$ $L_q = Tr(\{E_1, ..., E_q\})$, i.e., $L_m = Tr(\mathcal{H})$.*

It can be seen that the determination of $Tr(\mathcal{H})$ based on our algorithm does not depend on the order of $E_1, ..., E_m$.

Remark 2.1. (Demetrovics and Thi [7]). Denote $L_q = S_q \cup \{B_1, ..., B_{t_q}\}$, and $l_q (1 \leq q \leq m-1)$ be the number of elements of L_q. It can be seen that the worst-case time complexity of our algorithm is

$$\mathcal{O}(|R|^2 \sum_{q=0}^{m-1} t_q u_q),$$

where $l_0 = t_0 = 1$ and

$$u_q = \begin{cases} l_q - t_q, & \text{if } l_q > t_q; \\ 1, & \text{if } l_q = t_q. \end{cases}$$

Clearly, in each step of our algorithm L_q is a simple hypergraph. It is known that the size of arbitrary simple hypergraph over R cannot be greater than $C_n^{[n/2]}$, where $n = |R|$. $C_n^{[n/2]}$ is asymptotically equal to $2^{n+1/2}/(\pi.n)^{1/2}$. From this, the worst-case time complexity of our algorithm cannot be more than exponential in the number of attributes. In cases for which $l_q \leq l_m (q = 1, ..., m-1)$, it is easy to see that the time complexity of our algorithm is not greater than $\mathcal{O}(|R|^2 |\mathcal{H}||Tr(\mathcal{H})|^2)$. Thus, in these cases this algorithm finds $Tr(\mathcal{H})$ in polynomial time in $|R|, |\mathcal{H}|$ and $|Tr(\mathcal{H})|$. Obviously, if the number of elements of $\mathcal{H}$ is small, then this algorithm is very effective. It only requires polynomial time in $|R|$.

The above algorithm reminds that in [3], but its form seems to be more convenient for our applications.

The following proposition is obvious.

Proposition 2.1. (Demetrovics and Thi [7]). *The time complexity of finding $Tr(\mathcal{H})$ of a given hypergraph $\mathcal{H}$ is (in general) exponential in the number of elements of R.*

Proposition 2.1 is still true for a simple hypergraph.

However, if we restrict the number of edges of a hypergraph, then the time complexity of finding $Tr(\mathcal{H})$ of a given hypergraph $\mathcal{H}$ is polynomial time.

Algorithm 2.2.

Input: Let $\mathcal{H} = \{E_1, ..., E_k\}$ be a simple hypergraph over R, where k is a constant.

Output: $Tr(\mathcal{H})$.

Method:

Step 1: We construct the set

$$\mathcal{G} = \{\{e_1\} \cup ... \cup \{e_k\} : e_i \in E_i, 1 \leq i \leq k\}.$$

Step 2: Compute

$$m(\mathcal{G}) = \{E_i \in \mathcal{G} : \nexists E_j \in \mathcal{G} : E_j \subset E_i\}.$$

Step 3: Let $Tr(\mathcal{H}) = m(\mathcal{G})$.

It is obvious that $m(\mathcal{G}) = Tr(\mathcal{H})$. Furthermore, $\mathcal{G} \supseteq Tr(\mathcal{H})$, and $|\mathcal{G}| < |R|^k$. Hence, in this case Algorithm 2.2 finds $Tr(\mathcal{H})$ in polynomial time. Clearly, if k is small, then our algorithm is very effective.

Definition 2.3. *Let R be a set and $R' \subseteq R$ a subset of it. Then $\overline{R'}$ denotes $R - R'$. Let $\mathcal{H}$ be a hypergraph over R. Then $\overline{\mathcal{H}} = \{\overline{E} : E \in \mathcal{H}\}$ is called the comlemented hypergraph of $\mathcal{H}$.*

It is known [3] that if $\mathcal{H}$ is a hypergraph, then $\overline{\overline{\mathcal{H}}} = \mathcal{H}$, and $\mathcal{H}$ is simple iff $\overline{\mathcal{H}}$ is simple.

3 Boyce-Codd normal form and transversals

Definition 3.1. *Let $R = \{a_1, ..., a_n\}$ be a nonempty finite set of attributes. A functional dependency (FD) is a statement of form $X \to Y$, where $X, Y \subseteq R$. The FD $X \to Y$ holds in a relation $r = \{h_1, ..., h_m\}$ over R if*

$$(\forall h_i, h_j \in r)((\forall a \in X)(h_i(a) = h_j(a)) \Rightarrow (\forall b \in Y)(h_i(b) = h_j(b))).$$

We also say that r satisfies the FD $X \to Y$.

Let F_r be a family of all FDs that holds in r. Then $F = F_r$ satisfies

(F1) $X \to X \in F$,

(F2) $(X \to Y \in F, Y \to Z \in F) \Rightarrow (X \to Z \in F)$,

(F3) $(X \to Y \in F, X \subseteq V, W \subseteq Y) \Rightarrow (V \to W \in F)$,

(F4) $(X \to Y \in F, V \to W \in F) \Rightarrow (X \cup V \to Y \cup W \in F)$.

A family of FDs satisfying (F1) - (F4) is called a f-family over R.

Clearly, F_r is a f-family over R. It is known [1] that if F is an arbitrary f-family, then there is a relation r over R such that $F_r = F$.

Given a family F of FDs over R, there exists a unique minimal f-family F^+ that contains F. It can be seen that F^+ contains all FDs which can be derived from F by the rules (F1) - (F4).

A relation scheme s is a pair (R, F), where R is a set of attributes, and F is a set of FDs over R. Denote $X^+ = \{a \in R : X \to \{a\} \in F^+\}$. X^+ is called the closure of X over s. It is clear that, $X \to Y \in F^+$ iff $Y \subseteq X^+$.

Clearly, if $s = (R, F)$ is a relation scheme, then there is a relation r over R such that $F_r = F^+$ (see, [1]).

Let r be a relation, $s = (R, F)$ be a relation scheme over R and $A \subseteq R$. Then A is a key of r (a key of s) if $A \to R \in F_r (A \to R \in F^+)$. A is a minimal key of $r(s)$ if A is a key of $r(s)$ and any proper subset of A is not a key of $r(s)$.

Denote $K_r (K_s)$ the set of all minimal keys of $r(s)$. It can be seen that K_r, K_s are simple hypergraphs over R.

Definition 3.2. *Let $s = (R, F)$ be a relation scheme over R. We say that an attribute $a \in R$ is prime if it belongs to a minimal key of s, and nonprime otherwise. $s = (R, F)$ is in BCNF if $A \to \{a\} \notin F^+$ for $A^+ \neq R, a \notin A$.*

If a relation scheme is changed to a relation we have the definition of BCNF for relation.

Let s be a relation scheme and r a relation over R. We say that r represents s if $K_r = K_s$.

Definition 3.3. *Let r be a relation over R, and E_r the equality set of r, i.e. $E_r = \{E_{ij} : 1 \leq i < j \leq |r|\}$, where $E_{ij} = \{a \in R : h_i(a) = h_j(a)\}$. Let $T_r = \{E_{ij} \in E_r : \nexists E_{pq} \in E_r : E_{ij} \subset E_{pq}\}$. Then T_r is called the maximal equality system of r.*

Definition 3.4. *Let K be a simple hypergraph over R. We define the set of antikeys of K, denoted by K^{-1}, as follows:*

$$K^{-1} = \{A \subset R : (B \in K) \Rightarrow (B \not\subseteq A) \text{ and } (A \subset C) \Rightarrow (\exists B \in K)(B \subseteq C)\}.$$

It is easy to see that K^{-1} is also a simple hypergraph over R.

In this paper, we always assume that if a simple hypergraph plays the role of the set of minimal keys (antikeys), then this simple hypergraph is not empty (does not contain R).

Definition 3.5. *Let $s = (R, F)$ be a relation scheme and r a relation over R. For every $A \subseteq R$, set $I(A) = \{a \in R : A \rightarrow \{a\} \notin F^+\}$. Then $I(A)$ is called an independent set of s. For r, put $I(A) = \{a \in R : A \rightarrow \{a\} \notin F_r\}$. Denote by I_s the family of all independent sets of s.*

Set $m(s) = \{B \in I_s : B \neq \emptyset, \nexists C \in I_s : C \subset B\}$. $m(s)$ is called the family of all minimal independent sets of s. Clearly, $m(s)$ is a simple hypergraph over R.

It can be seen that A is a key of s if and only if $I(A) = \emptyset$.

Denote by I_r and $m(r)$ the family of all independent sets and the family of all minimal independent sets of r.

The following result was discovered in [7].

Theorem 3.1. (Demetrovics and Thi [7]). *Let $s = (R, F)$ be a relation scheme over R. Then*

$$Tr(K_s) = m(s).$$

It is known [3] that if $\mathcal{H}, \mathcal{G}$ are two simple hypergraphs over R, then $\mathcal{H} = Tr(\mathcal{G})$ if and only if $\mathcal{G} = Tr(\mathcal{H})$. From this we obtain

Corollary 3.1. *Let $s = (R, F)$ be a relation scheme over R. Then*

$$K_s = Tr(m(s)).$$

Definition 3.6. *Let $s = (R, F)$ be a relation scheme over R. We say that an attribute $a \in R$ is independent if it belongs to an independent set of s, and dependent otherwise.*

Denote by D_n the set of all dependent attributes of s. Clearly, $R - D_n$ is the set of all independent attributes of s.

Lemma 3.1. *Let $\mathcal{H}$ be a simple hypergraph over R. Then*

$$\cup Tr(\mathcal{H}) = \cup\mathcal{H}.$$

Proof. Assume that $a \in \cup Tr(\mathcal{H})$. Hence, there exists a minimal transversal T of $\mathcal{H}$ such that $a \in T$. From this, we obtain $a \in E, E \in \mathcal{H}$. This means that $a \in \cup\mathcal{H}$. Consequently, $\cup Tr(\mathcal{H}) \subseteq \cup\mathcal{H}$ holds.

Conversely, if $a \in \cup\mathcal{H}$ then there is $E \in \mathcal{H}$ such that $a \in E$. From this, according to the definition of transversal hypergraph of $\mathcal{H}$ there exists $T \in Tr(\mathcal{H})$ such that $a \in T$, i.e. $a \in \cup Tr(\mathcal{H})$. Hence, $\cup\mathcal{H} \subseteq \cup Tr(\mathcal{H})$. The proof is complete. □

From Lemma 3.1 we obtain the following

Corollary 3.2. *Let $s = (R, F)$ be a relation scheme over R, $m(s)$ be a family of all independent sets of s. Then*

$$\cup Tr(m(s)) = \cup m(s).$$

Theorem 3.2. *Let $s = (R, F)$ be a relation scheme over R. Then*

$$\cup K_s = R - D_n.$$

Proof. Assume that a is an element of $R - D_n$, i.e., there exists an $I(A) \in m(s)$ such that $a \in I(A)$. Hence, $a \in \cup m(s)$. By Corollary 3.2 we attain $a \in \cup Tr(m(s))$. By Theorem 3.1 we also obtain $a \in \cup K_s$. Thus, $R - D_n \subseteq \cup K_s$.

Conversely, suppose that $a \in \cup K_s$. Thus, by Corollary 3.1 and Corollary 3.2 $a \in \cup m(s)$. Hence, there exists an $I(A) \in m(s)$ such that $a \in I(A)$, i.e., $a \in R - D_n$. Consequently, $\cup K_s \subseteq R - D_n$.

The theorem is proved. □

Minimal keys and antikeys are related as follows:

Proposition 3.1. *Let $s = (R, F)$ be a relation scheme over R. Then*

$$K_s^{-1} = \overline{Tr(K_s)}.$$

Proof. Assume $X \in K_s^{-1}$. From Definition 3.4 we have that for every minimal key K, $K - X \neq \emptyset$, thus $\overline{X} \cap K \neq \emptyset$. Which implies that $\overline{X} \in Trs(K_s)$. On the other hand, according to the definition of antikey set, we have

$$X \cup \{a\} \supseteq K,$$

where $a \in \overline{X}$ and $K \in K_s$, which implies that $(\overline{X} - \{a\}) \cap K = \emptyset$. Consequently, $\overline{X} \in Tr(K_s)$, i.e., $X \in \overline{Tr(K_s)}$. Hence, we have $K_s^{-1} \subseteq \overline{Tr(K_s)}$.

Conversely, suppose that $Y \in Tr(K_s)$. Then $\overline{Y}$ is not superset of any minimal keys. Clearly, for all $a \in Y, Y - \{a\} \notin Trs(K_s)$, i.e. $(Y - \{a\}) \cap K = \emptyset$. This means that

$$\overline{Y} \cup \{b\} \supseteq K,$$

for all $b \in Y$. Consequently, $\overline{Tr(K_s)} \subseteq K_s^{-1}$.

The proposition is proved. □

Remark 3.1. Let $s = (R, F)$ be a relation scheme over R. Set $Z_s = \{A^+ : A \subseteq R\}$, i.e., Z_s is the set of all closures of s. Put $T_s = \{A \in Z_s : A \neq R, \nexists B \in Z_s : A \subset B\}$. Hence, T_s is the set of all maximal elements of $Z_s - \{R\}$. By the definition of the independent set of s, we can see that $T_s = \{R - B : B \in m(s)\}$.

From Theorem 3.1, Proposition 3.1 and Remark 3.1 we have

Proposition 3.2. *Let $s = (R, F)$ be a relation scheme over R. Then*

$$\overline{Tr(K_s)} = T_s.$$

The Proposition 3.2 means that for all $A \in \overline{Tr(K_s)} : A^+ = A$ and $A \neq R$.

Remark 3.2. Let r be a relation over R. From r we compute E_r. We construct the maximal equality system T_r of r. Then we have $T_r = K_r^{-1}$ (see, e.g., [8]). Denote elements of T_r by $A_1, ..., A_t$.

Set $M_r = \{B : B \neq \emptyset, B = A_i - \{a\} : a \in R, i = 1, ..., t\}$. Denote elements of M_r by $B_1, ..., B_l$. We construct a relation $r' = \{h_0, h_1, ..., h_l\}$ as follows:

$$\text{for all } a \in R, h_0(a) = 0, \forall i = 1, ..., l$$

$$h_i(a) = \begin{cases} 0, & \text{if } a \in B_i, \\ i, & \text{otherwise.} \end{cases}$$

Clearly, r' is in BCNF and $K_r = K_{r'}$.

We give the following algorithm that from a given relation scheme s constructs a relation r such that r represents s.

Algorithm 3.1.
Input: a BCNF relation scheme $s =< R, F >$.
Output: a BCNF relation r such that $K_r = K_s$.
Method:
Step 1: From s compute K_s.
Step 2: By Algorithm 2.1 we construct the set $Tr(K_s)$.
Step 3: Compute $\overline{Tr(K_s)}$. Denote elements of $\overline{Tr(K_s)}$ by $A_1, ..., A_t$.
Step 4: Set $Q_s = \{B : B \neq \emptyset, B = A_i - \{a\} : a \in R, i = 1, 2, ..., t\}$. Denote elements of Q_s by $B_1, ..., B_l$.
Step 5: Construct a relation $r = \{h_0, h_1, ..., h_l\}$ as follows:

$$\text{for all } a \in R, h_0(a) = 0, \forall i = 1, ..., l$$

$$h_i(a) = \begin{cases} 0, & \text{if } a \in B_i, \\ i, & \text{otherwise.} \end{cases}$$

Based on Proposition 3.1, Remark 3.2 and Proposition 3.2 we have $K_r = K_s$ and r is in BCNF. It is easy to see that the time complexity of Algorithm 3.1 is exponential in the number of attributes.

Let r be a relation over R. Let $N_r = \{N_{ij} : 1 \leq i < j \leq |r|\}$, where $N_{ij} = \{a \in R : h_i(a) \neq h_j(a)\}$. Then N_r is called the nonequality set of r.

Let $M_r = \{A \in N_r : \nexists B \in N_r : B \subset A\}$. M_r is called the minimal nonequality system of r.

The following result was discovered in [7].

Theorem 3.3. (Demetrovics and Thi [7]). *Let r be a relation over R. Then $K_r = Tr(M_r)$, where M_r is the minimal nonequality system of r.*

From Theorem 3.3 we have an effective application of Theorem 3.3, which is the following algorithm finding a BCNF relation scheme s such that $K_s = K_r$ from a given relation r in BCNF.

Algorithm 3.2.
Input: Let r be a BCNF relation over R.
Output: a BCNF relation scheme $s =< R, F >$ such that $K_s = K_r$.
Method:
Step 1: From r compute N_r.
Step 2: From N_r compute the minimal nonequality system M_r.
Step 3: By Algorithm 2.1 constructs $Tr(M_r)$. Clearly, $K_r = Tr(M_r)$.
Step 4: Denoting elements of K_r by $A_1, ..., A_m$. We construct a relation scheme as follows: $s =< R, F >$, where $F = \{A_1 \to R, ..., A_m \to R\}$.

Clearly, s is in BCNF and $K_s = K_r$. The time complexity of this algorithm is the time complexity of Algorithm 2.1. In many cases this algorithm is very effective (see Remark 2.1).

4 Special Sperner-systems and transversals

The notion of saturated Sperner-system is defined in [6] as follows:

Definition 4.1. (Demetrovics [6]). *A Sperner-system K over R is saturated if for any $A \subseteq R, K \cup \{A\}$ is not a Sperner-system.*

Now we are going to give a new characterization of saturate Sperner-systems. To do this, we need the following definition:

Definition 4.2. *Let $\mathcal{H}$ and $\mathcal{G}$ be two hypergraphs over R. Then $\mathcal{H} > \mathcal{G}$ iff for every $H \in \mathcal{H}$ there exists $G \in \mathcal{G}$ such that $H \supset G$, and $\mathcal{H} < \mathcal{G}$ iff for every $H \in \mathcal{H}$ there exists $G \in \mathcal{G}$ such that $H \subset G$.*

From this definition we obtain the following:

Proposition 4.1. *Let $\mathcal{H} \neq \emptyset$ and $\mathcal{G} \neq \emptyset$ be two hypergraphs over R. Then*
(1) $\emptyset > \mathcal{H}$ and $\emptyset < \mathcal{H}$.
(2) $\mathcal{H} > \{\emptyset\}$.
(3) $\{\emptyset\} < \mathcal{H}$.
(4) $\mathcal{H} > \mathcal{G}$ (resp. $\mathcal{H} < \mathcal{G}$) does not imply $\mathcal{G} < \mathcal{H}$ (resp. $\mathcal{G} > \mathcal{H}$.
(5) $\mathcal{H} < \{R\}$ iff $R \notin \mathcal{H}$.
(6) $\mathcal{H} \subseteq \mathcal{G}$ does not imply $\mathcal{H} < \mathcal{G}$.

Proof.
(1) It is obvious from Definition 4.2.
(2) Since $\mathcal{H}$ is hypergraph, we have (2).
(3) By similar arguments we also have (3).
(4) We give a counterexample. Let $R = \{a, b, c\}$. Consider the hypergraphs

$$\mathcal{H} = \{\{a, b\}\}, \mathcal{G} = \{\{a\}, \{b\}, \{c\}, \{a, b, c\}\}.$$

It holds that $\mathcal{H} > \mathcal{G}$ (resp. $\mathcal{H} < \mathcal{G}$), but it does not hold that $\mathcal{G} < \mathcal{H}$ (resp. $\mathcal{G} > \mathcal{H}$).
(5) From definition of hypergraphs and Definition 4.2 we obtain (5).
(6) We give a counterexample. Let $R = \{a, b\}$. Consider the hypergraphs

$$\mathcal{H} = \{\{a\}\}, \mathcal{G} = \{\{a\}, \{b\}\}.$$

It holds that $\mathcal{H} \subseteq \mathcal{G}$, but it does not hold that $\mathcal{H} < \mathcal{G}$.
The proposition is proved. □

Remark 4.1. $>$ and $<$ are transitive on the hypergraphs on R.

Theorem 4.1. *Let K be a Sperner-system over R. Then K is saturated if and only if $\overline{Tr(K)} < K$.*

Proof. Let K be a saturated Sperner-system. Suppose that there exists an $A \in \overline{Tr(K)}$ such that for every $B \in K, A \not\subset B$. By Proposition 3.1 and Definition 3.4 we have $K \cup \{A\}$, a Sperner-system. Which contradicts the hypothesis that K is saturated. Consequently, $\overline{Tr(K)} < K$.

Conversely, suppose that $\overline{Tr(K)} < K$, but K is not saturated. Hence, there exists an $A \subset R$ such that $K \cup \{A\}$ is a Sperner-system. Because $R \notin K$, for every $C \in K$ we have $C \subset R$. Thus, we can construct B such that $A \subseteq B$, $K \cup \{B\}$ is a Sperner-system and for every $D(B \subset D)$, there exists $C \in K$ such that $D \supseteq C$. Which implies that $B \in K^{-1}$. This contradicts the hypothesis $\overline{Tr(K)} < K$, i.e., for every $A \in K^{-1}$ (because $\overline{Tr(K)} = K^{-1}$), there exists $B \in K$ such that $A \subset B$. Consequently, K is saturated. The theorem is proved. □

Definition 4.3. *Let K be a Sperner-system over R. We say that K is embedded if for every $A \in K$ there is a $B \in H$ such that $A \subset B$, where $H^{-1} = K$.*

From Proposition 3.1, Theorem 4.1 we have the following

Proposition 4.2. *Let K be a Sperner-system over R. Then K is saturated if and only if $\overline{Tr(K)}$ is embedded.*

From Proposition 3.1 and Proposition 4.2 the following corollary is immediate:

Corollary 4.1. *Let K be a Sperner-system over R. Then K is saturated if and only if K^{-1} is embedded.*

Corollary 4.1 was shown in [12].

Definition 4.4. *Let K be a Sperner-system over R. We say that K is inclusive if for every $A \in K$, there exists a $B \in K^{-1}$ such that $B \subset A$.*

From Proposition 3.1, Definition 4.2 and Definition 4.4, the following proposition is evident.

Proposition 4.3. *Let K be a Sperner-system over R. Then K is inclusive if and only if $K > \overline{Tr(K)}$.*

Remark 4.2. (Demetrovics [4]). If K is an arbitrary Sperner-system over R, then there is a relation scheme $s = (R, F)$ such that $K = K_s$.

Theorem 4.2. *Let K be a Sperner-system over R. Then K is inclusive if and only if $\overline{Tr(\overline{Tr(K)})} < \overline{Tr(K)}$.*

Proof. Suppose that K is an inclusive Sperner-system, but there exists an $A \in \overline{Tr(\overline{Tr(K)})}$ such that for every $B \in \overline{Tr(K)}, A \not\subset B$. Hence, $\overline{Tr(K)} \cup \{A\}$ is a Sperner-system. By Remark 4.2, for K there is a relation scheme s such that $K = K_s$. If $A^+ \subset R$ then according to Proposition 3.2 there exists $C \in \overline{Tr(K)}$ such that $A^+ \subseteq C$, which contradicts the fact that $\overline{Tr(K)} \cup \{A\}$ is a Sperner-system. Consequently, A is a key of s. It is obvious that there is a minimal key $A'(A' \subseteq A)$ such that $A' \in K$. Thus, $\overline{Tr(K)} \cup \{A'\}$ is a Sperner-system. By Proposition 4.3, this is a contradiction. Consequently, $\overline{Tr(\overline{Tr(K)})} < \overline{Tr(K)}$.

Conversely, assume that $\overline{Tr(\overline{Tr(K)})} < \overline{Tr(K)}$. By Proposition 4.2, we obtain which $\overline{Tr(K)}$ is saturated. From this, Proposition 3.2 and Proposition 4.3, we have K, an inclusive Sperner-system. The theorem is proved. □

From Theorem 4.2, Definition 4.3, Proposition 4.2 and Proposition 3.1, we have the following

Corollary 4.2. *K is an inclusive Sperner-system if and only if K^{-1} is a saturated one.*

Corollary 4.2 was shown in [12].

From Corollary 4.1 and Corollary 4.2 the following corollary is obvious:

Corollary 4.3. *Let K be a Sperner-system over R. Denote H a Sperner-system for which $H^{-1} = K$. Then the followings are equivalent:*

(1) K is saturated,

(2) K^{-1} is embedded,

(3) H is inclusive.

References

[1] Armstrong W. W., *Dependency Structure of Database Relationship*, Information Processing 74, North-Holland Pub. Co. (1974) 580-583.

[2] Beeri C., Bernstein P. A., *Computational Problems Related to the Design of Normal Form Relation Scheme*, ACM Trans. on Database Syst. **4, 1** (1979) 30-59.

[3] Berge C., *Hypergraphs: Combinatorics of Finite Sets*, North - Holland, Amsterdam (1989).

[4] Demetrovics J.,*Logical and structural investigation of relation datamodel*, MTA SZTAKI Tanulm-ányok, **114** (1980), 1-97 (in Hungarian).

[5] Demetrovics J., *On the equivalence of candidate keys with Sperner systems*, Acta Cybernetica **4**, (1979), 247-252.

[6] Demetrovics J., F üredi Z., Katona G., *A függőségek és az individumok száma közötti kapcsolat összetett adatrendszerek esetén*, Alkalmazott Matematikai Lapok **9** (1983),13-21.

[7] Demetrovics J., Thi V. D., *Describing Candidate Keys by Hypergraphs*, Computers and Artificial Intelligence **18, 2** (1999), 191-207.

[8] Demetrovics J., Thi V.D., *On the time complexity of algorithms related to Boyce-Codd normal form*, SERDICA-Bulgaricae Mathematicae Publicationes, **19**, (1993), 134-144.

[9] Demetrovics J., Thi V. D., *Some computational problems related to Boyce-Codd normal form*, Annales Univ. Sci. Budapest. Sect. Comp. **19**, (2000), 119-132.

[10] Gottlob G., Libkin L., *Investigations on Armstrony relations, denpendency inference, and excluded functional dependencies*, Acta Cybernetica Hungary **9, 4** (1990), 385-402.

[11] Lucchesi C. L., Osborn S. L., *Candidate keys for relations*, J. Comput. Syst. Scien. **17, 2** (1978), 270-279.

[12] Thi V. D., *Minimal keys and Antikeys*, Acta Cybernetica **7, 4** (1986), 361-371.

Received January, 2004

Acta Cybernetica **16** (2004) 485–506.

Relationships Between Closure Operations and Choice Functions – Equivalent Descriptions of a Family of Functional Dependencies

Nghia D. Vu*

Abstract

The family of functional dependencies plays an important role in the relational database. The main goal of this paper is to investigate closure operations and choice functions. They are equivalent descriptions of family of functional dependencies. The main properties of and relationship between closure operations and choice functions are presented in this paper.

1 Introduction

The motivation of this study is equivalent descriptions of family of functional dependencies (FDs). FDs play a significant role in the implementations of relational database model, which was defined by E.F Codd. However, relational database is still one of the most powerful databases. One of the most important branches in the theory of relational database is that dealing with the design of database schemes. This branch is based on the theory of FDs and constraints. Armstrong observed that FDs give rise to closure operations on the set of attributes. And he shows that closure operation is an equivalent description of family of FDs, that is, the family of all FDs satisfying Armstrong axiom stated in next section. That the family of FDs can be described by closure operations on the attributes' set plays a very important role in theory of relational database. Because this representation was successfully applied to find many properties of FDs, studying those properties of closure operations is indirect way of finding that of the family of FDs. Besides closure operations, there are some other representations of family of FDs. Such as, the closed sets of a closure form a semilattice. And the semilattice with greatest elements gives an equivalent description of FDs. The closure operations, and other equivalent descriptions of family of FDs have been studied widely by Armstrong [Ar], Beeri, Dowd, Fagin and Statman [BDFS], and H. Mannila and K.J.Raiha [MR]. More, see [DK2], [DHLM], [DT3], and [Li]. Studying equivalent descriptions of family of FDs helps us to understand deeper the family FDs and widens the

*Institute of Information Technology of Vietnam, Department of Database Management System, 18 Hoang Quoc Viet, Hanoi, Vietnam, email: nghiavu@cse.buffalo.edu

study of it. Closure operation is widely known and considered the representation of family of functional dependencies most studied. Among equivalent descriptions of functional dependencies, the properties of choice functions are not developed well enough in contrast to those of closure operations. Moreover, a closure operation can be derived from a choice function and vice versa. Thus, by studying properties of choice function satisfying reverse inclusion was studied in connection with the theory of rational behavior of individuals and groups. For the study on choice function and relationship between closure operations and choice functions, see [DHLM] and [Li].

For relation schemes $s =< U, F >$ and $t =< U, V >$, where U is a set of attributes and F and V sets of FDs over U, we are always able to build a closure $L_1(A)$ on F, for every A is a set of attributes on U. However, if we build $L_2(L_1(A))$ on V, we find out that a meet-semilattice can not be formed from this computation. That is, we can not form a relation scheme from this computation. We are going to show in this paper what condition that provide to build the composition $L_1(L_2(A))$ such that a relation scheme can be formed from this composition. In other words, what is necessary and sufficient conditions that make sure $L_1(L_2(A))$ is a closure. We find this result through the studies of choice functions. Besides that, many properties of choice functions will be studies in depth. The interaction of choice functions and closure operations also are investigated widely in this paper. We also study the relationship between choice functions and FDs. Those results can be used to build many algorithm problems related to choice functions and closure operation and family of FDs.

Direct product of decomposition of a closure operation plays an important role in the theory and practice of relational database. If we consider a relation of database as a matrix, a row contains the data of one individual, the estimation of the minimum cardinality of rows of such matrix is very valuable in practice of relational database. The studies of estimation of the minimum cardinality of rows for direct product of decomposition of a closure operation can be found variously in [DFK], [Li], [DK2]. In this paper we present the new notion and properties of direct product of decomposition of choice function.

In the next section some necessary definitions and facts about relational database, some equivalent descriptions of family of functional dependencies besides choice function and closure operation theory are given.

The result of this paper is presented in the third section. They are organized into six parts as follows.

Part 1 represents necessary and sufficient condition of composition of choice functions to be a choice function. The studies of composition of closure operations has been shown through those of choice functions. The main result of this paper will be presented in depth in Part 1.

The direct product of decomposition of a choice function is in Part 2.

It will be proposed in Part 3 to study some fundamental properties of a composition of closure operations and choice functions. We are giving important properties of intersection, union, and composition of choice functions, which will be fully investigated in depth in Part 1.

In Part 4, we show relationship between and interactive properties of closure operations and choice functions. In this section we consider the closure for which choice function defined in next section satisfies some additional properties.

In Part 5, we are presenting a class of special choice functions, which is very useful in the studying of combinatorial problems related to choice functions and closure operations.

Part 6 gives some special relationship between choice functions and family of FDs, which helps us intensively into algorithm problems on building choice functions and closure operations. Since the theoretical result presented here are preliminary. Thus many open problems in the studies of choice functions and closure operation will be shown in this paper.

2 Basic Definitions

Let us give some formal definitions that are used in the next sections. Those well-known concepts in relational database given in this section can be found in [Ar, BB, BDFS, DK2, DT1, and Ul].

A relational database system of the scheme $R(a_1, ..., a_n)$ is considered as a table, where columns correspond to the attributes ai 's while the row are $n-$tuples of relation r. Let X and Y be nonempty sets of attributes in R. We say that instance r of R satisfies the FD if two tuples agree on the values in attributes X, they must also agree on the values in attributes Y. Here is the formal mathematical definition of FDs.

Definition 2.1. *Let $U = \{a_1, ..., a_n\}$ be a nonempty finite set of attributes. A functional dependency is a statement of the form $A \to B$, where $A, B \subseteq U$. The FD A B holds in a relation $R = \{h_1, ..., h_m\}$ over U if $\forall h_i, h_j \in R$ we have $h_i(a) = h_j(a)$ for all $a \in A$ implies $h_i(b) = h_j(b)$ for all $b \in B$. We also say that R satisfies the FD $A \to B$.*

Let F_R be a family of all FDs that hold in R.

Definition 2.2. *Then $F = F_R$ satisfies.*

(1) $A \to A \in F$,

(2) $(A \to B \in F, B \to C \in F) \Longrightarrow (A \to C \in F)$,

(3) $(A \to B \in F, A \subseteq C, D \subseteq B) \Longrightarrow (C \to D \in F)$,

(4) $(A \to B \in F, C \to D \in F) \Longrightarrow (A \cup C \to B \cup D \in F)$.

A family of FDs satisfying (1)-(4) is called an f-family over U.

Clearly, F_R is an f-family over U. It is known [Ar] that if F is an arbitrary f-family, then there is a relation R over U such that $F_R = F$.

Given a family F of FDs over U, there exits a unique minimal f-family F^+ that contains F. It can be seen that F^+ contains all FDs which can be derived from F by the rules (1)-(4).

Definition 2.3. *A relation scheme s is a pair $< U, F >$, where U is a set of attributes, and F is a set of FDs over U.*

Denote $A^+ = \{a : A \to \{a\} \in F^+\}$. A^+ is called the closure of A over s. It is clear that $A \to B \in F^+$ iff $B \subseteq A^+$. Clearly, if $s =< U, F >$ is a relation scheme, then there is a relation R over U such that $F_R = F+$ (see, [Ar]).

Definition 2.4. *Let U be a nonempty finite set of attributes and $P(U)$ its power set. A map $L : P(U) \to P(U)$ is called a closure operation (closure for short) over U if it satisfies the following conditions:*

(1)$A \subseteq L(A)$, (Extensiveness Property)

(2)$A \subseteq B$ implies $L(A) \subseteq L(B)$, (Monotonicity Property)

(3) $L(L(A)) = L(A)$. (Closure Property)

Let $s =< U, F >$ be a relation scheme. Set $L(A) = \{a : A \to \{a\} \in F^+\}$, we can see that L is a closure over U.

Theorem 2.1. *[Ar] If F is a f-family and if $L_F = \{a : a \in U \text{ and } A \to \{a\} \in F\}$, then L_F is a closure. Inversely, if L is a closure, there exists only a f-family F over U such that $L = L_F$, and $F = \{A \to B : A, B \subseteq U, B \subseteq L(A)\}$.*

Let $L \subseteq P(U)$. L is called a meet-irreducible family over U (sometimes it is called a family of members which are not intersection of two other members) if $A, B, C \in L$, then $A = BC$ implies $A = B$ or $A = C$.

Let $I \subseteq P(U), U \in I$, and $A, B \in I \Rightarrow A \cap B \in I$. I is called a meet-semilattice over U. Let $M \subseteq P(U)$.

Denote $M^+ = \{\cap M' : M' \subseteq M\}$. We say that M is a generator of I if $M^+ = I$. Note that $U \in M^+$ but not in M, by convention it is the intersection of the empty collection of sets. Denote $N \doteq \{A \in I : A \neq \cap\{A' \in I : A \subset A'\}\}$. In [DK2] it is proved that N is the unique minimal generator of I.

It can be seen that N is a family of members which are not intersections of two other members.

Let L be a closure operation over U. Denote $Z(L) = \{A : L(A) = A\}$ and $N(L) = \{A \in Z(L) : A \neq \cap\{A' \in Z(L) : A \subset A'\}\}$. $Z(L)$ is called the family of closed sets of L. We say that $N(L)$ is the minimal generator of L.

It is shown [DK2] that if N is a meet-irreducible family then there is a closure L such that N is the minimal generator of it.

Theorem 2.2. *[Ar] There is an on-to-one correspondence between meet- irreducible families and f-families on U.*

Theorem 2.3. *[DK2] There is a 1-1 correspondence between meet-irreducible families and meet-semilattices on U.*

Definition 2.5. *Let $M \subseteq P(U)$. M is called a Sperner system over U if $A, B \in M$, then A is not a subset of B.*

Definition 2.6. *Let U be a nonempty finite set of attributes. A family $M = \{(A, \{a\}) : A \subset U, a \in U\}$ is called a maximal family of attributes over R iff the following conditions are satisfied:*

(1) $a \notin A$,

(2) For all $(B, \{b\}) \in M, a \notin B$ *and* $A \subseteq B$ *imply*$A = B$.

(3) $\exists (B, \{b\}) \in M : a \notin B, a \neq b$, *and* $L_a \cup B$ *is a Sperner system over* R, *where* $L_a = \{A : (A, \{a\}) \in M\}$.

Remark 2.1.

- It is possible that there are $(A, \{a\}), (B, \{b\}) \in M$ such that $a \neq b$, but $A = B$.
- It can be seen that by (1) and (2) for each$a \in U, L_a$ is a Sperner system over U. It is possible that L_a is an empty Sperner system.
- Let U be a nonempty finite set of attribute and $P(U)$ its power set. According to Definition 2.6 we can see that given a family $Y \subseteq P(U) \times P(U)$ there is a polynomial time algorithm deciding whether Y is a maximal family of attribute over U.

Let L be a closure over R. Denote $Z(L) = \{A : L(A) = A\}$ and $M(L) = \{(A, \{A\}) : A \notin A, A \in Z(L)$ and $B \in Z(L), A \subseteq B, A \notin B$ imply $A = B\}$.

$Z(L)$ is called the family of closed sets of L. It can be seen that for each $(A, \{a\}) \in M(L)$.Ais a maximal closed set which doesn't contain a.

It is possible that there are $(A, \{a\}), (B, \{b\}) \in M(L)$ such that $a \neq b$, but $A = B$.

The following theorem which shows that closure operations and maximal families of attributes determine each other uniquely.

Theorem 2.4. *[DT4] LetL be a closure operation over* U. *Then* $M(L)$ *is a maximal family of attributes over* U. *Conversely, if* M *is a maximal family of attributes overU, then there exists exactly one closure operation* L *over* U *so that* $M(L) = M$, *where for all* $B \in P(U)$

$$H(B) = \begin{cases} \bigcap_{B \subseteq A} A \text{ if } \exists A \in L(M) : B \subseteq A, \\ R \text{ otherwise,} \end{cases}$$

and $L(M) = \{a : (a, \{a\}) \in M\}$.

Now, we introduce the following concept.

Definition 2.7. *LetY* $\in P(U) \times P(U)$. *We say that* Y *is a minimal family overU if the following conditions are satisfied:*

(1) $\forall (A, B), (A', B') \in Y : A \subset B \subseteq U, A \subset A'$ *implies* $B \subset B', A \subset B'$ *implies* $B \subseteq B'$,

(2) Put $U(Y) = \{B : (A, B) \in Y\}$. *For each* $B \in U(Y)$ *and* C *such that* $C \subset B$ *and there is no* $B' \in U(Y) : C \subset B' \subset B$, *there is an* $A \in L(B) : A \subseteq C$, *where* $L(B) = \{A : (A, B) \in Y\}$.

Remark 2.2.

- $U \in U(Y)$.
- From $A \subset B'$ implies $B \subseteq B'$ there is no a $B' \in U(Y)$ such that $A \subset B' \subset B$ and $A = A'$ implies $B = B'$.
- Because $A \subset A'$ implies $B \subset B'$ and $A = A'$ implies $B = B'$, we can be see that $L(B)$ is a Sperner system over R and by (2) $L(B) \neq \emptyset$.

Let I be a meet-semilattice over R. Put $M^*(I) = \{(A, B) : \exists C \in I$ such that $A \subset C, A \neq \cap\{C : C \in I, A \subset C\}, B = \cap\{C : C \in I, A \subset C\}\}$. Set $M(I) = \{(A, B) \in M^*(I)$: there does not exist $(A', B) \in M^*(I)$ such that $A' \subset A\}$.

Theorem 2.5. *[DT4] Let I be a meet-semilattice over U. Then$M(I)$is a minimal family over$U$. Conversely, if Y is a minimal family over U, then there is exactly one meet-semilattice I so that $M(I) = Y$, where $I = \{C \subseteq R : \forall (A, B) \in Y : A \subseteq C$ implies $B \subseteq C\}$.*

Let Z be an intersection semilattice on U and suppose that $H \subset U, H \notin Z$hold and $Z \cup \{H\}$ is also closed under intersection. Consider the sets A satisfying $A \in Z, H \subset A$. The intersection of all of these sets is in Z therefore it is different form H. Denote it by $L(H)$. $H \subset L(H)$ is obvious. Let $H(Z)$ denote the set of all pairs $(H, L(H))$ where $H \subset U, H \notin Z$, but $Z \cup \{H\}$ is closed under intersection. The following theorem characterize the possible sets $H(Z)$:

Theorem 2.6. *[DK1] The set $\{(A_i, B_i)|i = 1, ..., m\}$ is equal to $H(Z)$ for some intersection semilattice Z iff the following conditions are satisfied:*

$A_i \subset B_i \subseteq U, A_i \neq B_i$,

$A_i \neq A_j$ *implies either* $B_i \subseteq A_j$, *or* $A_j \subseteq B_i$,

$A_i \subseteq B_j$ *implies* $B_i \subseteq B_j$,

for any i and $C \subset U$ satisfying $A_i \subset C \subset B_i (A_i \neq C \neq B_i)$.
There is a j such that either $C = A_j$ or $A_j \subset C, B_j \not\subset C, C \not\subset B_j$ all hold.

The set of pair (A_i, B_i) satisfying those condition above is called an extension. Its definition is not really beautiful but it is needed in some application. On the other hand it is also an equivalent notion to the closures:

Theorem 2.7. *[DK1] $Z \to H(Z)$ is a bijection between the set of intersection semilattices and the set of extensions.*

Definition 2.8. *Let U be a nonempty finite set of attributes and $P(U)$ its power set. A map $C : P(U) \to P(U)$ is called a choice function, if every $A \in P(U)$, then $C(A) \subseteq A$.*

U is interpreted as a set of alternatives, A as a set of alternatives given to the decision-maker to choose the best and $C(A)$ as a choice of the best alternatives among A.

Let L be a closure operation, we define C and H associated with L as follows:

$$C(A) = U - L(U - A), \tag{*}$$

and

$$H(A) = A \cap L(U - A). \tag{**}$$

We can easily prove that $C(A)$ and $H(A)$ are two choice functions. And we name $C(A)$ choice function - I (for short, CF-I), and $H(A)$ choice function - II (for short, CF-II).

Theorem 2.8. *The relationship like (*) is considered as a 1-1 correspondence between closures and choice functions, which satisfies the following two conditions: For every $A, B \subseteq U$,*

(1) If $C(A) \subseteq B \subseteq A$, then $C(A) = C(B)$ (Out Casting Property),

(2) If $A \subseteq B$, then $C(A) \subseteq C(B)$ (Monotonicity Property).

Theorem 2.9. *The relationship like (**) is considered as a 1-1 correspondence between closures and choice functions, which satisfies the following two conditions: For every $A, B \subseteq U$,*

(1) If If $H(A) \subseteq B \subseteq A$, then $H(A) = H(B)$ (Out Casting Property),

(2) If $A \subseteq B$, then $H(B) \cap A \subseteq H(A)$ (Heredity Property).

We also note that both C and H uniquely determine the closure L as the following

$$L(A) = U - C(U - A) \text{ and } H(A) = A \cup L(U - A).$$

For every $A \subseteq U$, the sets $C(A)$ and $H(A)$ form a partition of A, that is, $C(A) \cup H(A) = A$, and $C(A) \cap H(A) = \emptyset$.

Theorem 2.10. *There is a 1-1 correspondence between CFs - I and closure operations on U.*

Theorem 2.11. *There is a 1-1 correspondence between CFs - II and closure operations on U.*

3 Results

3.1 Properties of and Relationship between Composition of Closure Operations and CFs - I and - II

First of all, we are giving the formal definition of composition of functions.

Definition 3.1. *Let f and g be two functions (e.g. closure operations, CFs - I, or II) on U, and we determine a map T as a composition of f and g the following:*

$$T(X) = f(g(X)) = f.g(X) = fg(X) \text{ for every } X \subseteq U.$$

In this section we are going to answer two questions. The first one is: given many CFs - I (or II), what can be said about the composition of those CFs - I (or II). In other words, what is necessary and sufficient conditions that provide that composition to be a CF - I (or II). The second one is: what is the relationship between that composition of CFs - I(or II) and that of closure operations. And if we can find the necessary and sufficient conditions that provide that composition of closure operations to be closure operation through those of CFs - I(or II).

With the same questions, however, first we are going to investigate problems with two choice functions. For convenient, we show the results of CFs - II. We will soon see that

Theorem 3.1. *Let H_1 and H_2 be CFs-II on U, then composition H_1H_2 and H_2H_1 are a CFs-II on U, and $H_1H_2 = H_2H_1 = H_1 \cap H_2$.*

However, to achieve this result, we necessarily prove those following Lemmas and Propositions. First we need to prove the following Proposition:

Proposition 3.1. *Let H_1 and H_2 be CFs - II on U, then for all $X \subseteq U$,*

$$H_1(X) \cap H_2(X) \text{ is a CF - II on } U.$$

To prove $H_1 \cap H_2$ is a CF - II, we need to prove the following.

Lemma 3.1. *Let L_1 and L_2 be closure operations on U, then for all $X \subseteq U$,*

$$L_1(X) \cap L_2(X) \text{ is a closure operation on } U.$$

Proof. Assume L_1 and L_2 be two closure operations on U, then for all $X \subseteq U$, it is easy to obtain that $X \subseteq L_1(X) \cap L_2(X)$ since $X \subseteq L_1(X)$ and $X \subseteq L_2(X)$. Now, to prove the Monotonicity Property of $L_1(X) \cap L_2(X)$, for every $X \subseteq Y$, we have $L_1(X) \subseteq L_1(Y)$ and $L_2(X) \subseteq L_2(Y)$. Therefore, $L_1(X) \cap L_2(X) \subseteq L_1(Y) \cap L_2(Y)$, so $L_1 \cap L_2$ satisfies Monotonicity Property. Then, we have to prove Closure Property of $L_1 \cap L_2$. We always have $X \subseteq L_1(X) \cap L_2(X) \subseteq L_1(X)$. Using Monotonicity Property of L_1, we attain $L_1(X) \subseteq L_1(L_1(X) \cap L_2(X)) \subseteq L_1(L_1(X)) = L_1(X)$. That means $L_1(X) = L_1(L_1(X) \cap L_2(X))$. Similarly, we attain that $L_2(X) = L_2(L_1(X) \cap L_2(X))$. Therefore, $L_1(X) \cap L_2(X) = L_1(L_1(X) \cap L_2(X)) \cap (L_2(L_1(X) \cap L_2(X))$. That is, $L_1 \cap L_2$ satisfies Closure Property, so $L_1 \cap L_2$ is a closure on U. The proof is completed. □

Now we are moving on proving Proposition 3.1.

Proof of Proposition 3.1. Assume H_1 and H_2 be CFs - II on U, then for all $X \subseteq U$, we have $H_1(X) = X \cap L_1(U - X)$, and $H_2(X) = X \cap L_2(U - X)$, with L_1 and L_2 two closure operations corresponding to H_1 and H_2 respectively. Thus $H_1(X) \cap H2(X) = (X \cap L_1(U-X)) \cap (X \cap L_2(U-X)) = X \cap L_1(U-X) \cap L_2(U-X)$. However, due to Lemma 3.1, $L_1(U-X) \cap L_2(U-X)$ is a closure operation, that is, there exists a closure operation L_3 such that $L_3(U-X) = L_1(U-X) \cap L_2(U-X)$. Thus, $C_1(X) \cap C_2(X) = X \cap L_3(U-X) = C_3(X)$, with C_3 is a CF - II corresponding to L_3. The proof is completed. □

Before proving Theorem 3.1, we need to prove the follows.

Lemma 3.2. *Let H_1 and H_2 be CFs - II on U, then*
1) $H_1H_2 = H_2H_1H_2$.
2) $H_2H_1 = H_1H_2H_1$

Proof. Assume H_1 and H_2 be CFs - II on U. Then for all $X \subseteq U, H_1(X) = X \cap L_1(U - X)$ and $H_2(X) = X \cap L_2(U - X)$, with L_1 and L_2 two closure operations corresponding to H_1 and H_2 respectively. $H_1H_2(X) = H_1(H_2(X)) = X \cap L_2(U - X) \cap L_1(U - X \cap L_2(U - X)) \subseteq X$. Due to Heredity Property of CFs - II for H_2, we obtain $H_2(X) \cap H_1H_2(X) \subseteq H_2(H_1H_2(X))$. By using $H_1H_2(X) = H_1(H_2(X)) \subseteq H_2(X)$, we attain $H_1H_2(X) \subseteq H_2(H_1H_2(X)) \subseteq H_1H_2(X)$. Hence $H_1H_2(X) = H_2(H_1H_2(X))$, that is, $H_1H_2 = H_2H_1H_2$. Similarly, we obtain $H_2H_1 = H_1H_2H_1$.The proof is completed. □

Lemma 3.3. *Let H_1 and H_2 be CFs - II on U, then following is equivalence:*
1)$H_1 \subseteq H_2$
2)$H_1H_2 = H_1$

Proof.
$(1 \rightarrow 2)$. Assume H_1 and H_2 be CFs-II on U and $H_1 \subseteq H_2$. Since H_1 is a CF-II, H_1 must satisfy Out Casting property: if $H_1(X) \subseteq Y \subseteq X$, then $H_1(X) = H_1(Y)$. Therefore, we have $H_1 \subseteq H_2$ or $H_1(X) \subseteq H_2(X) \subseteq X$ for every $X \subseteq U$, so $H_1(H_2(X)) = H_1(X)$ or we conclude that $H_1H_2 = H_1$.

$(2 \rightarrow 1)$. Assume H_1 and H_2 be CFs - II on U and $H_1H_2 = H_1$. Since H_1 and H_2 are CFs - II, according to Definition of choice function, we have $H_1H_2 \subseteq H_2$, but $H_1H_2 = H_1$, so we have $H_1 \subseteq H_2$. The proof is completed. □

Easily, we obtain the following Corollary.

Corollary 3.1. *If H is a CF - II on U, then $HH = H$.*

Proof of Theorem 3.1. Assume H_1 and H_2 be CFs - II on U. Then for all $X \subseteq U, H_2(X) \subseteq X$. Due to Heredity Property of CF - II for H_1, we obtain $H_1(X) \cap H_2(X) \subseteq H_1(H_2(X))$. Besides that, $H_1(H_2(X)) \subseteq H_2(X) \subseteq X$, we obtain $H_1 \cap H_2(X) \subseteq H_1H_2(X) \subseteq X$. By Proposition 3.1, $H_1(X) \cap H_2(X)$ is a CF - II. Using

Out Casting Property for $H_1 \cap H_2$, we achieve $H_1 \cap H_2(H_1H_2(X)) = H_1 \cap H_2(X)$ or $H_1(H_1H_2(X)) \cap H_2(H_1H_2(X)) = H_1 \cap H_2(X)$. Due to Corollary 3.1, we obtain $H_1(H_1H_2(X)) = H_1H_2(X)$, and Lemma 3.2, we obtain $H_1H_2(X) = H_2H_1H_2(X)$. Therefore, we attain that $H_1H_2(X) = H_1 \cap H_2(X)$, that is $H_1H_2 = H_1 \cap H_2$.That means H_1H_2 is a CF-II. Similarly, we obtain $H_2H_1 = H_1 \cap H_2$ and H_2H_1 is a CF-II. The proof is completed. □

We can generalize Theorem 3.1 by the following

Generalization 3.1. Let H_i be CFs - II on U with $i = 1 \rightarrow n$, then $H_{i1}H_{i2}....H_{i(n-1)}H_{in}$ is a CFs - II on U, and

$$H_{i1}H_{i2}...H_{in} = \bigcap_{i=1}^{n} H_i$$

with $\{H_{i1}, H_{i2}., ..., H_{i(n-1)}, H_{in}\}$ be permutations of $\{H_1, H_2, .., H_{(n-1)}, H_n\}$.

Thus, for CFs - II, a composition of CFs - II is always a CF - II. Now we move on the composition of CFs - I before investigating on closure operations.

Theorem 3.2. *Let C_1 and C_2 be CFs - I on U. A composition of C_1 and C_2, denoted as C_1C_2, is a CF - I if and only if*

$$C_1C_2C_1 = C_1C_2.$$

However, before proving Theorem 3.2, we need to have the following Lemmas.

Lemma 3.4. *Let C_1 and C_2 be CF-Is on U. Then*

1) $C_1C_2 \subseteq C_1$,
2) $C_1C_2 \subseteq C_2$,
3) $C_2C_1 \subseteq C_1$,
4) $C_2C_1 \subseteq C_2$.

Due to Definition of choice function, and Monotonicity Property of CFs - II, clearly we obtain that Lemma.

Lemma 3.5. *Let C_1 and C_2 be CFs - I on U, then following is equivalence:*

1) $C_1 \subseteq C_2$,
2) $C_1C_2 = C_1$.

Proof. The proof of this Lemma is similar to that of Lemma 3.3. □

Easily, we obtain the following Corollary.

Corollary 3.2. *If C is a CF - I on U, then $CC = C$.*

Now we move on proving Theorem 3.2.

Proof Theorem 3.2. Assume C_1 and C_2 be CFs-I on U and the composition C_1C_2 also a CF - I. Due to Lemma 3.4, we have $C_1C_2(X) \subseteq C_1(X) \subseteq X$. Due to Out Casting Property of composition C_1C_2, we have $C_1C_2C_1(X) = C_1C_2(X)$.

Inversely, assume C_1 and C_2 be CFs-I on U and the composition C_1C_2 satisfying that $C_1C_2C_1 = C_1C_2$. For all $X \subseteq U$, it is clear to obtain that $C_1(C_2(X)) \subseteq C_2(X) \subseteq X$. It means C_1C_2 is a choice function. Now, to prove the Monotonicity Property of the composition C_1C_2. For every $X \subseteq Y$, using Monotonicity Property of C_1 and C_2, we have $C_2(X) \subseteq C_2(Y)$, then $C_1(C_2(X)) \subseteq C_1(C_2(Y))$. That means that C_1C_2 satisfies Monotonicity Property. Then, we have to prove Out Casting Property of the composition C_1C_2. For all X and $Y \subseteq U, C_1C_2(X) \subseteq Y \subseteq X$, we need to prove that $C_1C_2(X) = C_1C_2(Y)$. Using Monotonicity Property of C_1, we obtain that $C_2(C_1C_2(X)) \subseteq C_2(Y) \subseteq C_2(X)$. Applying Monotonicity Property of C_1 once again, we have $C_1(C_2C_1C_2(X)) \subseteq C_1(C_2(Y)) \subseteq C_1(C_2(X))$. However, $C_1C_2C_1 = C_1C_2$. Therefore, $C_1C_2C_2(X) \subseteq C_1C_2(Y) \subseteq C_1C_2(X)$. Due to Corollary 3.2, we obtain that $C_1C_2(X) \subseteq C_1C_2(Y) \subseteq C_1C_2(X)$ That means $C_1C_2(X) = C_1C_2(Y)$. That is, C_1C_2 satisfies Out Casting Property, so C_1C_2 is a CF-I on U. The proof is completed. □

We generalize the Theorem above.

Generalization 3.2. Let $C_1, C_2, ..,$ and C_n be CFs-I on U. A composition of $C_1, C_2, ...,$and C_n, denoted as $C_1C_2....C_{n-1}C_n$, is a CF-I if and only if

$$C_1C_2...C_{n-1}C_nC_1C_2....C_{n-1} = C_1C_2...C_{n-1}C_n.$$

Proof. We prove this Generalization by induction. It is obvious for $n = 1$. The Theorem 3.2 proves the case that $n = 2$.

For $n = k$, we assume that $C_1, C_2, ..,$ and C_k be CFs-I on U, and the composition of $C_1, C_2, ...,$ and C_k, denoted as $C_1C_2....C_{k-1}C_k$, is a CF - I. We need to prove that, for $n = k + 1$, the composition $C_1C_2....C_kC_{k+1}$ is a CF - I iff $C_1C_2...C_kC_{k+1}C_1C_2....C_k = C_1C_2...C_kC_{k+1}$. Surely, since $C_1C_2....C_{k-1}C_k$ is a CF - I, by using Theorem 3.2, we obtain that the composition $C_1C_2....C_kC_{k+1}$ is a CF - I iff $C_1C_2...C_kC_{k+1}C_1C_2....C_k = C_1C_2...C_kC_{k+1}$. The proof is completed. □

Now we move on the relationship between the composition of closure operations and CFs - I. We have the following Theorem.

Theorem 3.3. *Let L_1 and L_2 be closure operations and C_1 and C_2 be CF-Is corresponding to L_1 and L_2 respectively on U. The following are equivalent:*

1) C_1C_2 is a CF-I,

2)L_1L_2 is a closure operation.

Proof. $(1 \rightarrow 2)$. Assume L_1 and L_2 be closure operations, and C_1 and C_2 be CF-Is corresponding to L_1 and L_2 respectively on U and C_1C_2 is a closure operation. Then for all $X \subseteq U$, we have $L_1(X) = U - C_1(U - X)$, and $L_2(X) = U - C_2(U - X)$. Thus, $L_1L_2(X) = L_1(L_2(X)) = U - C_1(U - (U - C_2(U - X))) = U - C_1(C_2(U - X)) = U - C_1C_2(U - X)$. However, C_1C_2 is a closure operation. Therefore,

there exists a closure operation C_3 such that $C_3(U-X) = C_1C_2(U-X)$. Thus, $L_1L_2(X) = U - C_3(U-X) = L_3(X)$, with L_3 a closure operation corresponding to C_3. That is L_1L_2 is a closure operation. The proof is completed.

$(2 \rightarrow 1)$. Assume L_1 and L_2 be closure operations and C_1 and C_2 be CF-Is corresponding to L_1 and L_2 respectively on U and L_1L_2 is a closure operation. Then for all $X \subseteq U$, we have $C_1(X) = U - L_1(U-X)$, and $C_2(X) = U - L_2(U-X)$. Thus, $C_1C_2(X) = C_1(C_2(X)) = U - L_1(U - (U - L_2(U-X))) = U - L_1(L_2(U-X)) = U - L_1L_2(U-X)$. However, L_1L_2 is a closure operation. Therefore, there exists a closure operation L_3 such that $L_3(U-X) = L_1L_2(U-X)$. Thus, $C_1C_2(X) = U - L_3(U-X) = C_3(X)$, with C_3 a choice function-I corresponding to L_3. That is C_1C_2 is a CF-I. The proof is completed. □

It can be seen that.

Generalization 3.3. Let L_i be closure operations and $\{C_i\}$ be CFs - I corresponding to L_i respectively on U, with $i = 1 \rightarrow n$. The following are equivalent:

1) $L_1L_2...L_n$ is a closure operation

2) $C_1C_2...C_n$ is a CF-I

And we also have $L_1L_2...L_n(X) = U - C_1C_2...C_n(U-X)$ and $C_1C_2...C_n(X) = U - L_1L_2...L_n(U-X)$.

Through Theorem 3.2 and 3.3, it is easy to obtain the following Theorem.

Theorem 3.4. *Let L_1 and L_2 be closure operations on U. A composition of L_1 and L_2, denoted as L_1L_2, is a closure operation if and only if*

$$L_1L_2L_1 = L_1L_2.$$

For generalization, we have the same conclusion for following Generalization as Generalization 3.2.

Generalization 3.4. Let $L_1, L_2, ..,$ and L_n be closure operations on U. A composite function of $L_1, L_2, ...,$ and L_n, denoted as $L_1L_2....L_{n-1}L_n$, is a closure operation if and only if

$$L_1L_2...L_{n-1}L_nL_1L_2....L_{n-1} = L_1L_2...L_{n-1}L_n.$$

3.2 Direct Product of CFs - I and - II

The direct product of closure operations plays very important role in theory of relational database, especially in combinatorial problems. Plenty of properties related to direct product of closure operation can be found in [DFK] and [Li]. By relationship and interaction between closure operations and choice functions, we introduce the new definitions of direct product of choice function-Is as well as -IIs. First of all, we have the following.

Theorem 3.5. *[Li] Let L_1 and L_2 be closure operations on the disjoint ground sets U_1 and U_2 respectively. The direct product of closure operations $L_1 \times L_2$ is defined as following*

$$(L_1 \times L_2)(X) = L_1(X \cap U_1) \cup L_2(X \cap U_2), \qquad X \subseteq U_1 \cup U_2.$$

Then $(L_1 \times L_2)(X)$ *is a closure operations on* $U_1 \cup U_2$.

Here we give the Generalization of above Theorem.

Generalization 3.5. Let $\{L_i |\ i = 1 \to n\}$ be closure operations on the disjoint ground sets U_i respectively. The direct product of those closure operations $L_1 \times L_2 \times ... \times L_n$ is defined as following

$$(L_1 \times L_2 \times \ldots \times L_n)(X) = \bigcup_{i=1}^{n} L_i(X \cap U_i)$$

with $X \subseteq U_1 \cup U_2 \cup ... \cup U_n$.

Then $(L_1 \times L_2 \times ... \times L_n)(X)$ is a closure operation on $U_1 \cup U_2 \cup ... \cup U_n$.

Theorem 3.6. *Let* C_1 *and* C_2 *be CFs - I on the disjoint ground sets* U_1 *and* U_2 *respectively. The direct product of CFs - I,* $C_1 \times C_2$, *is defined as following*

$$(C_1 \times C_2)(X) = C_1(X \cap U_1) \cup C_2(X \cap U_2), \qquad X \subseteq U_1 \cup U_2.$$

Then $(C_1 \times C_2)(X)$ *is a CF - I on* $U_1 \cup U_2$.

Proof. For all $X \subseteq U_1 \cup U_2, (C_1 \times C_2)(X) = C_1(X \cap U_1) \cup C_2(X \cap U_2) \subseteq (X \cap U_1) \cup (X \cap U_2) \subseteq X \cap (U_1 \cup U_2) = X$. Thus, $(C_1 \times C_2)(X) \subseteq X$. For every X and $Y \subseteq U_1 \cup U_2$ and $X \subseteq Y$, then $X \cap U_1 \subseteq Y \cap U_1$, and $X \cap U_2 \subseteq Y \cap U_2$. By using Monotonicity Property of C_1 and C_2, we obtain $C_1(X \cap U_1) \subseteq C_1(Y \cap U_1)$ and $C_2(X \cap U_2) \subseteq C_2(Y \cap U_2)$. Hence $C_1(X \cap U_1) \cup C_2(X \cap U_2) \subseteq C_1(Y \cap U_1) \cup C_2(Y \cap U_2)$, that is, $(C_1 \times C_2)(X) \subseteq (C_1 \times C_2)(Y)$ or $(C_1 \times C_2)$ satisfies Monotonicity Property. Now we need to show that $(C_1 \times C_2)(X)$ satisfies the Out Casting Property also. That is, for every $X, Y \subseteq U_1 \cup U_2$ and $(C_1 \times C_2)(X) = C_1(X \cap U_1) \cup C_2(X \cap U_2) \subseteq Y \subseteq X$, we need to show that $(C_1 \times C_2)(X) = (C_1 \times C_2)(Y)$. Since $Y \subseteq X$, we have $(C_1 \times C_2)(Y) \subseteq (C_1 \times C_2)(X)$. And it is obvious that $C_1(X \cap U_1) \subseteq C_1(X \cap U_1) \cup C_2(X \cap U_2) \subseteq Y$. Thus, we have $C_1(X \cap U_1) \cap U_1 \subseteq Y \cap U_1$ or $C_1(X \cap U_1) \subseteq Y \cap U_1$. Using Monotonicity Property of C_1, we have $C_1(C_1(X \cap U_1)) \subseteq C_1(Y \cap U_1)$ or $C_1(X \cap U_1) \subseteq C_1(Y \cap U_1)$ due to Corollary 3.2. Similarly, we obtain $C_2(X \cap U_1) \subseteq C_2(Y \cap U_1)$. Therefore $C_1(X \cap U_1) \cup C_2(X \cap U_2) \subseteq C_1(Y \cap U_1) \cup C_2(Y \cap U_2)$ or $(C_1 \times C_2)(X) \subseteq (C_1 \times C_2)(Y)$. Hence $(C_1 \times C_2)(X) = (C_1 \times C_2)(Y)$. The proof is completed. □

Generalization 3.6. Let $\{C_i |\ i = 1 \to n\}$ be CFs - I with on the disjoint ground sets $\{U_i\}$ respectively. The direct product of CFs - I, $C_1 \times C_2 \times ... \times C_n$, is defined as following

$$(C_1 \times C_2 \times \ldots \times C_n)(X) = \bigcup_{i=1}^{n} C_i(X \cap U_i)$$

with $X \subseteq U_1 \cup U_2 \cup ... \cup U_n$.

Then $(C_1 \times C_2 \times ... \times C_n)(X)$ is a CF - I on $U_1 \cup U_2 \cup ... \cup U_n$.

Theorem 3.7. *Let H_1 and H_2 be CFs - II on the disjoint ground sets U_1 and U_2 respectively. The direct product of CFs - II, $H_1 \times H_2$, is defined as following*

$$(H_1 \times H_2)(X) = H_1(X \cap U_1) \cup H_2(X \cap U_2), \qquad X \subseteq U_1 \cup U_2.$$

Then $(H_1 \times H_2)(X)$ is a CF - II on $U_1 \cup U_2$.

Proof. For all $X \subseteq U_1 \cup U_2, (H_1 \times H_2)(X) = H_1(X \cap U_1) \cup H_2(X \cap U_2) \subseteq (X \cap U_1) \cup (X \cap U_2) \subseteq X \cap (U_1 \cup U_2) = X$. Thus, $(H_1 \times H_2)(X) \subseteq X$. For every X and $Y \subseteq U_1 \cup U_2$ and $X \subseteq Y$, we need to prove that $(H_1 \times H_2)$ satisfies Heredity Property. Since $X \subseteq Y$, we have $X \cap U_1 \subseteq Y \cap U_1$, and $X \cap U_2 \subseteq Y \cap U_2$. By using Heredity Property of H_1 and H_2, we obtain $H_1(Y \cap U_1) \cap (X \cap U_1) \subseteq H_1(X \cap U_1)$ or $H_1(Y \cap U_1) \cap X \subseteq H_1(X \cap U_1)$. Similarly, we have $H_2(Y \cap U_2) \cap X \subseteq H_2(X \cap U_2)$. Hence, $(H_1(Y \cap U_1) \cap X) \cup (H_2(Y \cap U_2) \cap X) \subseteq H_1(X \cap U_1) \cup H_2(X \cap U_2)$, then $(H_1(Y \cap U_1) \cup H_2(Y \cap U_2)) \cap X \subseteq H_1(X \cap U_1) \cup H_2(X \cap U_2)$ that is, $(H_1 \times H_2)(Y) \cap X \subseteq (H_1 \times H_2)(X)$ or $(H_1 \times H_2)$ satisfies Heredity Property.

Now we need to show that $(H_1 \times H_2)(X)$ satisfies the Out Casting Property also. That is, for every X and $Y \subseteq U_1 \cup U_2$ and $(H_1 \times H_2)(X) = H_1(X \cap U_1) \cup H_2(X \cap U_2) \subseteq Y \subseteq X$, we need to show that $(H_1 \times H_2)(X) = (H_1 \times H_2)(Y)$. It is obvious that $H_1(X \cap U_1) \subseteq H_1(X \cap U_1) \cup H_2(X \cap U_2) \subseteq Y \subseteq X$. Then $H_1(X \cap U_1) \cap U_1 \subseteq Y \cap U_1 \subseteq X \cap U_1$ or $H_1(X \cap U_1) \subseteq Y \cap U_1 \subseteq X \cap U_1$. Using Out Casting Property of H_1, we obtain $H_1(X \cap U_1) = H_1(Y \cap U_1)$. Similarly, we attain $H_2(X \cap U_2) = H_2(Y \cap U_2)$. Therefore $H_1(X \cap U_1) \cup H_2(X \cap U_2) = H_1(Y \cap U_1) \cup H_2(Y \cap U_2)$ or $(H_1 \times H_2)(X) = (H_1 \times H_2)(Y)$. The proof is completed. □

Generalization 3.7. Let $\{H_i |\ i = 1 \to n\}$ be CFs - II with on the disjoint ground sets U_i respectively. The direct product of CFs - II, $H_1 \times H_2 \times ... \times H_n$, is defined as following

$$(H_1 \times H_2 \times \ldots \times H_n)(X) = \bigcup_{i=1}^{n} H_i(X \cap U_i)$$

with $X \subseteq U_1 \cup U_2 \cup ... \cup U_n$.

Then $(H_1 \times H_2 \times \ldots \times H_n)(X)$ is a CF - II on $U_1 \cup U_2 \cup ... \cup U_n$.

3.3 Properties of CFs - I and - II and Closure Operations

Proposition 3.2. *Let C_1 and C_2 be CFs-I on U, then for all $X \subseteq U$,*

$$C_1(X) \cup C_2(X) \text{ is a CF-I on } U.$$

Proof. Assume C_1 and C_2 be CFs-I on U, then for all $X \subseteq U$, it is easy to obtain that $C_1(X) \cup C_2(X) \subseteq X$ since $C_1(X) \subseteq X$ and $C_2(X) \subseteq X$. Now, to prove the Monotonicity Property of $C_1 \cup C_2$, for every $X \subseteq Y$, we have $C_1(X) \subseteq C_1(Y)$ and $C_2(X) \subseteq C_2(Y)$. Therefore, $C_1(X) \cup C_2(X) \subseteq C_1(Y) \cup C_2(Y)$, so $C_1 \cup C_2$ satisfies Monotonicity Property. Then, we have to prove Out Casting Property of $C_1 \cup C_2$. We always have $C_1(X) \subseteq C_1(X) \cup C_2(X) \subseteq Y \subseteq X$. Using Out Casting Property

of C_1, we attain $C_1(X) = C_1(Y)$. Similarly, we attain that $C_2(X) = C_2(Y)$ from $C_2(X) \subseteq C_1(X) \cup C_2(X) \subseteq Y \subseteq X$. Therefore, $C_1 \cup C_2(X) = C_1 \cup C_2(Y)$. That is, $C_1 \cup C_2$ satisfies Out Casting Property, so $C_1 \cup C_2$ is a CF-I on U. The proof is completed. □

Proposition 3.3. *Let H_1 and H_2 be CFs-II on U, then for all $X \subseteq U$,*

$$H_1(X) \cup H2(X) \text{ is a CF-II on } U.$$

Proof. Assume H_1 and H_2 be CFs-II on U. Similarly to above proof, for all $X \subseteq U$ it is clear to obtain that $H_1(X) \cup H_2(X) \subseteq X$ since $H_1(X) \subseteq X$ and $H_2(X) \subseteq X$. Now, to prove the Heredity Property of $H_1 \cup H_2$, for every $X \subseteq Y$, we have $H_1(Y) \cap X \subseteq H_1(X)$ and $H_2(Y) \cap X \subseteq H_2(X)$. Therefore, $X \cap (H_1(Y) \cup H_2(Y)) \subseteq H_1(X) \cup H_2(X)$, so $H_1 \cup H_2$ satisfies Heredity Property. For Out Casting Property of $H_1 \cup H_2$, we prove the same as the proof of Proposition 3.2. The proof is completed. □

From Proposition 3.3, we lead to the following Lemmas.

Lemma 3.6. *Let L_1 and L_2 be closure operations on U, then for all $X \subseteq U$,*

$$L_1(X) \cup L_2(X) \text{ is a closure operation on } U.$$

Proof. Assume L_1 and L_2 be closure operations on U, then for all $X \subseteq U$, we have $L_1(X) = X \cup H_1(U - X), L_2(X) = X \cup H_2(U - X)$, with H_1 and H_2 two choice function-IIs corresponding to L_1 and L_2 respectively. Thus $L_1(X) \cup L_2(X) = X \cup H_1(U-X) \cup H_2(U-X)$. However, due to Proposition 3.3, $H_1(U-X) \cup H_2(U-X)$ is a CF - II, that is, there exists a choice function H_3 such that $H_3(U - X) = H_1(U - X) \cup H_2(U - X)$. Thus, $L_1(X) \cup L_2(X) = X \cup H_3(U - X) = L_3(X)$, with L_3 a closure operation corresponding to H_3. The proof is completed. □

Using similar method of above proof, we can achieve two following.

Lemma 3.7. *Let C_1 and C_2 be CFs - I on U, then for all $X \subseteq U$,*

$$C_1(X) \cap C_2(X) \text{ is a CF - I on } U.$$

Proof. Assume C_1 and C_2 be CFs - I on U, then for all $X \subseteq U$, we have $C_1(X) = U - L_1(U-X)$, and $C_2(X) = U - L_2(U-X)$, with L_1 and L_2 two closure operations corresponding to C_1 and C_2 respectively. Thus $C_1(X) \cap C_2(X) = (U - L_1(U - X)) \cap (U - L_2(U-X)) = U - L_1(U-X) \cup L_2(U-X)$. However, due to Lemma 3.6, $L_1(U-X) \cup L_2(U-X)$ is a closure operation, that is, there exists a closure operation L_3 such that $L_3(U - X) = L_1(U - X) \cup L_2(U - X)$. Thus, $C_1(X) \cup C_2(X) = U - L_3(U - X) = C_3(X)$, with C_3 a CF - I corresponding to L_3. The proof is completed. □

Proposition 3.4. *Let H be a CF-II on U. Then for all $X \subseteq U$, we have*

$$H(X) \cap H(Y) \subseteq H(X \cap Y).$$

Proof. For all X and $Y \subseteq U$, due to Monotonicity Property of closure operations, we easily obtain $L(X) \cap L(Y) \subseteq L(X \cup Y)$. Therefore, $L(U - X) \cap L(U - Y) \subseteq L((U - X) \cup (U - Y))$. Using $L((U - X) \cup (U - Y)) = L(U - X \cap Y)$, we have $L(U - X) \cap L(U - Y) \subseteq L(U - X \cap Y)$. Hence, $(X \cap Y) \cap L(U - X) \cap L(U - Y) \subseteq (X \cap Y) \cap L((U - X \cap Y)$ or $H(X) \cap H(Y) \subseteq H(X \cap Y)$. The proof is completed. □

Similarly, we obtain the follow

Proposition 3.5. *Let H be a CF-II on U. Then for all $X \subseteq U$, we have $H(X \cup Y) \subseteq H(X) \cup H(Y)$.*

Proof. For all X and $Y \subseteq U$, due to Monotonicity Property of closure operations, we easily obtain $L(X \cap Y) \subseteq L(X) \cap L(Y)$. Therefore, $L((U - X) \cap (U - Y)) \subseteq L(U - X) \cap L(U - Y)$. Using $L((U - X) \cap (U - Y)) = L(U - X \cup Y)$, we have $L(U - X \cup Y) \subseteq L(U - X) \cap L(U - Y)$. Hence, $(X \cup Y) \cap L((U - X \cup Y) \subseteq (X \cup Y) \cap L(U - X) \cap L(U - Y)$ or $H(X \cap Y) \subseteq (X \cap L(U - X) \cap L(U - Y)) \cup (Y \cap L(U - X) \cap L(U - Y)) \subseteq (X \cap L(U - X)) \cup (Y \cap L(U - Y)) = H(X) \cup H(Y)$. The proof is completed. □

Lemma 3.8. *Let H_1 and H_2 be CFs-II on U. Then*
1) $H_1H_2 \subseteq H_2$
2) $H_2H_1 \subseteq H_1$

Since H_1 and H_2 are a CFs-II , it is obvious to have above Lemma.

Lemma 3.9. *Let H_1 and H_2 be CFs - II on U, then*
1) $H_1 \cap H_2 \subseteq H_1H_2$
2) $H_1 \cap H_2 \subseteq H_2H_1$

Proof. Assume H_1 and H_2 be CFs - II on U. Then for all $X \subseteq U, H_2(X) \subseteq X$. Due to Heredity Property of CFs-II, we obtain $H_1(X) \cap H_2(X) \subseteq H_1(H_2(X))$, that is, $H_1 \cap H_2 \subseteq H_1H_2$. Similarly, we achieve $H_1 \cap H_2 \subseteq H_2H_1$. □

Proposition 3.6. *Let H_1 and H_2 be CFs - II on U, then $H_1 \cap H_2 = H_1 \cap H_1H_2 = H_2 \cap H_2H_1$.*

In order to prove this Proposition, we need to have the following Lemma.

Lemma 3.10. *Let H_1 and H_2 be CFs - II on U, then $H_1 \cap H_2 = H_1(H_1 \cap H_2) = H_2(H_1 \cap H_2)$.*

Proof. Assume H_1 and H_2 be CFs - II on U. Then for all $X \subseteq U$, we always have $H_1(X) \cap H_2(X) \subseteq H_2(X)$. Due to Heredity Property of CF-IIs, we obtain $H_1(H_2(X)) \cap H_1(X) \cap H_2(X) \subseteq H_1(H_1(X) \cap H_2(X))$. According to Lemma 3.9, we obtain $H_1(X) \cap H_2(X) \subseteq H_1(H_1(X) \cap H_2(X))$. However, $H_1(H_1(X) \cap H_2(X)) \subseteq H_1(X) \cap H_2(X)$. Hence, $H_1(H_1(X) \cap H_2(X)) = H_1(X) \cap H_2(X)$, that is, $H_1 \cap H_2 = H_1(H_1 \cap H_2)$. Similarly, we achieve $H_1 \cap H_2 = H_2(H_1 \cap H_2)$. The proof is completed. □

Proof of Proposition 3.6. Assume H_1 and H_2 be CFs - II on U. For all $X \subseteq U$ due to Proposition 3.4 and Corollary 3.1, we obtain $H_1(X) \cap H_1(H_2(X)) \subseteq H_1(H_1(X) \cap H_2(X))$. However, $H_1 \cap H_2 = H_1(H_1 \cap H_2)$ according to Lemma 3.10, and $H_1 \cap H_2 \subseteq H_1H_2$ due to Lemma 3.9. Therefore, $H_1(X) \cap H_1(X) \cap H_2(X) \subseteq H_1(X) \cap H_1(H_2(X)) \subseteq H_1(X) \cap H_2(X)$. Then, $H_1(X) \cap H_1(H_2(X)) = H_1(X) \cap H_2(X)$, that is, $H_1 \cap H_2 = H_1 \cap H_1H_2$. Similarly, we obtain $H_1 \cap H_2 = H_2 \cap H_2H_1$. The proof is completed. □

From Proposition 3.6, it is clear to obtain the follow.

Corollary 3.3. *Let H_1 and H_2 be CFs - II on U, then $H_1 \cap H_2 = H_1 \cap H_2(H_1 \cap H_2)$.*

3.4 Interaction between Closure Operations and CFs - I

Let L be a closure and Σ a corresponding full family of FDs. We recall that an FD $X \to Z \in \Sigma$ iff $Z \subseteq L(X)$. In this section, we consider the closures for which CF - I and -II defined in section 0 satisfy some additional properties. We are now going to give some properties.

Proposition 3.7. *Let L and C be a closure operation and a CF-I corresponding to Lrespectively on U. The following are equivalent:*

1) $C(X \cup Y) = C(X) \cup C(Y)$,
2) $L(X \cap Y) = L(X) \cap L(Y)$,
3) $X \to Z$ and $Y \to Z$ are FDs from Σ iff $X \cap Y \to Z$.

Proof. (1 → 2). Let C satisfies 1). Then for all $X, Y \subseteq U$: $L(X \cap Y) = U - C(U - X \cap Y) = U - C((U - X) \cup (U - Y)) = U - C(U - X) \cup C(U - Y) = (U - C(U - X)) \cap (U - C(U - Y)) = L(X) \cap L(Y)$. That is, L satisfies 2).

(2 → 1) Let L satisfies 2). Then for all $X, Y \subseteq U$: $C(X \cup Y) = U - L(U - X \cup Y) = U - L((U - X) \cap (U - Y)) = U - L(U - X) \cap L(U - Y) = (U - L(U - X)) \cup (U - L(U - Y)) = C(X) \cup C(Y)$. That is, C satisfies 1).

(2 ↔ 3) Let L satisfies 2). Then for all $X, Y \subseteq U$: $L(X \cap Y) = L(X) \cap L(Y)$. For $Z \in L(X \cap Y)$ iff $X \cap Y \to Z$. And $Z \in L(X) \cap L(Y)$, that means $Z \in L(X)$ and $Z \in L(Y)$ iff $X \to Z$ and $Y \to Z$. □

Proposition 3.8. *Let L and C be a closure operation and a CF-I corresponding to L respectively on U. The following are equivalent:*

1) $C(X \cap Y) = C(X) \cap C(Y)$,
2) $L(X \cup Y) = L(X) \cup L(Y)$.

Proof. (1 → 2). Let C satisfies 1). Then for all $X, Y \subseteq U$: $L(X \cup Y) = U - C(U - X \cup Y) = U - C((U - X) \cap (U - Y)) = U - C(U - X) \cap C(U - Y) = (U - C(U - X)) \cup (U - C(U - Y)) = L(X) \cup L(Y)$. That is, L satisfies 2).

(2 → 1) Let L satisfies 2). Then for all $X, Y \subseteq U$: $C(X \cap Y) = U - L(U - X \cap Y) = U - L((U - X) \cup (U - Y)) = U - L(U - X) \cup L(U - Y) = (U - L(U - X)) \cap (U - L(U - Y)) = C(X) \cap C(Y)$. That is, C satisfies 1). □

Proposition 3.9. *Let L_1 and L_2 be closure operations and C_1 and C_2 be CF-Is corresponding to L_1 and L_2 respectively on U. The following are equivalent:*

1) $C_1(X) \cap C_2(X) \subseteq C_1C_2(X)$

2) $L_1L_2(X) \subseteq L_1(X) \cup L_2(X)$

Proof. $(1 \rightarrow 2)$. Let C_1 and C_2 satisfy 1). Then for all $X \subseteq U$: $L_1L_2(X) = U - C_1C_2(U-X) \subseteq U - C_1(U-X) \cap C_2(U-X) = (U - C_1(U-X)) \cup (U - C_2(U-X)) = L_1(X) \cup L_2(X)$. That is, L_1 and L_2 satisfy 2).

$(2 \rightarrow 1)$. Let L_1 and L_2 satisfy 2). Then for all $X \subseteq U$: $C_1(X) \cap C_2(X) = (U - L_1(U-X)) \cap (U - L_2(U-X)) = U - L_1(U-X) \cup L_2(U-X) \subseteq U - L_1L_2(U-X) = C_1C_2(X)$. That is, C_1 and C_2 satisfy 1). □

3.5 Special cases of Choice Function-Is and -IIs

Theorem 3.8. *Let consider a partition $V : \{V_1, V_2, V_3, ..., V_n\}$, that is, $V_i \cap V_j = \emptyset$, with $i \neq j$. Let construct a set*

$$W(A) = A \cap \bigcup_{i=1}^{n} V_i$$

for all $A \subseteq U$. Then, $W(A)$ is a CF-I on U.

Proof. For all $A \subseteq U$, it is clear that $W(A) \subseteq A$. Now we need to prove that W satisfies Monotonicity and Out Casting Property. We have

$$W(A) = A \cap \bigcup_{i=1}^{n} V_i = \bigcup_{i=1}^{n} (A \cap V_i)$$

$$\Rightarrow W(W(A)) = \bigcup_{j=1}^{n} (A \cap \bigcup_{i=1}^{n} V_i) \cap V_j = \bigcup_{j=1}^{n} (\bigcup_{i=1}^{n} (A \cap V_i \cap V_j))$$

$$= \bigcup_{i=1}^{n} (A \cap V_i) = W(A),$$

since $V_i \cap V_j = \emptyset$, for $i \neq j$. For $A \subseteq B$, it is obvious that $A \cap V_i \subseteq B \cap V_i$, then

$$\bigcup_{i=1}^{n} (A \cap V_i) \subseteq \bigcup_{i=1}^{n} (B \cap V_i).$$

Thus, $W(A) \subseteq W(B)$, so W satisfies Monotonicity Property.

To prove Out Casting Property of W, let assume $W(A) \subseteq B \subseteq A$, we have show that $W(A) = W(B)$. Using Monotonicity Property of W, we attain $W(W(A)) \subseteq W(B) \subseteq W(A)$. However, $W(W(A)) = W(A)$, we lead to that $W(A) = W(B)$. The proof is completed. □

We can illustrate $W(A)$ as the sum of all intersections of A and V_i , for $i = 1 \rightarrow n$. Here is a property of W.

Proposition 3.10. *Let consider partition of* $V : \{V_1, V_2, V_3..., V_n\}$, *that is,* $V_i \cap V_j = \emptyset$, *with* $i \neq j$, *and partition of* $T : \{T_1, T_2, T_3..., T_m\}$, *that is,* $T_i \cap T_j = \emptyset$, *with* $i \neq j$. *For all* $A \subseteq U$, *let construct two CF-I as the following:*

$$C_1(A) = A \cap \bigcup_{i=1}^{n} V_i,$$

$$C_2(A) = A \cap \bigcup_{j=1}^{m} T_j.$$

Then, $C_1(A) \cap C_2(A) = C_1C_2(A)$, *and both also are CF-Is.*

Proof. For all $A \subseteq U$, we have

$$C_1(A) \cap C_2(A) = (A \cap \bigcup_{i=1}^{n} V_i) \cap (A \cap \bigcup_{j=1}^{m} T_j) = A \cap (\bigcup_{i=1}^{n} V_i \cap \bigcup_{j=1}^{m} T_j)$$
$$= (A \cap \bigcup_{j=1}^{m} T_j) \cap \bigcup_{i=1}^{n} V_j = C_1C_2(A).$$

However,

$$C_1(A) \cap C_2(A) = A \cap (\bigcup_{i=1}^{n} V_i \cap \bigcup_{j=1}^{m} T_j) = A \cap \bigcup_{i=1}^{n} (\bigcup_{j=1}^{m} T_j \cap V_i).$$

It is easy to see that, for every $x \neq y$,

$$(\bigcup_{j=1}^{m} T_j \cap V_x) \cap (\bigcup_{j=1}^{m} T_j \cap V_y) = \emptyset.$$

That is, $\{(\bigcup T_j \cap V_i) | i = 1 \rightarrow n, j = 1 \rightarrow m\}$ is a partition. Due to Theorem 3.10, we conclude that $C_1(A) \cap C_2(A)$ as well as $C_1C_2(A)$ is a CF-I. The proof is completed. □

Let us define $W_c(A)$, the complementary set of $W(A)$, as $W_c(A) = A - W(A)$, that is

$$W_c(A) = A - A \cap \bigcup_{i=1}^{n} V_i = (A - A) \cup (A - \bigcup_{i=1}^{n} V_i) = A - \bigcup_{i=1}^{n} V_i = \bigcap_{i=1}^{n} (A - V_i).$$

Since $W(A)$ is a CF-I, and CF-I and CF-II of A form a partition of A, for every $A \subseteq U$, we lead to the following Theorem.

Theorem 3.9. *Let consider partition of* $V : \{V_1, V_2, V_3..., V_n\}$, *that is,* $V_i \cap V_j = \emptyset$, *with* $i \neq j$. *Let construct a set*

$$W_c(A) = \bigcap_{i=1}^{n} (A - V_i)$$

for all $A \subseteq U$. *Then,* $W_c(A)$ *is a CF-II on* U.

3.6 Discussion and Open Problems

Given a set of F of functional dependencies over U and the attribute set $X \subseteq U$, so the functional dependencies closure of $X, L(X)$, is the set $\{A \subseteq U | X \rightarrow A \in F\}$. It turns out that this set is independent of the underlying attribute set U. We have known that two types of choice function -I and -II associated with L as follows:

$$C(A) = U - L(U - A), \text{ and} H(A) = A \cap L(U - A).$$

Thus, given a set of F of functional dependencies, we define, $X \subseteq U$, choice-I and -II of X as follows:

$$H_F(X) = X \cap \{A \subseteq U | (U - X) \rightarrow A \in F\} \quad (1)$$

$$C_F(X) = U - \{A \subseteq U | (U - X) \rightarrow A \in F\} \quad (2)$$

It can be seen the following Propositions.

Proposition 3.11. *Let* F *be a set of functional dependencies and* $X \rightarrow Y$ *an functional dependency. Then* $X \rightarrow Y \in F$ *iff* $Y \not\subset C_F(U - X)$.

Proposition 3.12. *Let* F *be a set of functional dependencies and* $X \rightarrow Y$ *an functional dependency. Then* $X \rightarrow Y \in F$ *and* $Y \not\in X$ *iff* $Y \subseteq H_F(U - X)$.

Now we move to compute $C_F(X)$ and $H_F(X)$. First of all, we now mention about the Algorithm of computing a closure from a set of functional dependencies and X a set of attributes.

In [BB], we were known the Algorithm to computing closure operation, by using relation between choice functions and closure operation, we can easily build Algorithm to compute choice functions.

Even though we already have an algorithm to compute closure of X, from the Theorem 3.4 above as follows: Let L_1 and L_2 be closure operations on U. A composite function of L_1 and L_2, denoted as L_1L_2, is a closure operation if and only if

$$L_1L_2L_1 = L_1L_2.$$

Open problems are set up as following:

Open Problem 1. Let $s =< U, F >$ and $t =< U, V >$ two relation schemes, where U is a set of attributes and F and V are two different sets of FDs over U. We define F^+ and V^+ be a set of all FDs that can be derived from F and V respectively.

1) Is it possible build a closure L_1 and a closure L_2 from F^+ and V^+ respectively such that $L_1L_2 = L_1L_2L_1$?

2) If so, how can we design L_1L_2 ? In other word, how can we design a relation scheme $w =< U, H >$ from which we can build H^+ , from which we can design the closure $L_1L_2 = L_1L_2L_1$?

3) If so, is it possible to generalize this design for more than two closure operations?

Open Problem 2. A similar problem as above, but for choice -I and -II of X.

Open Problem 3. Algorithm problems related to union and intersection for choice -I and -II and closures.

Open Problem 4. Generalize those theories presented in this paper to mutilvalued dependencies.

References

[Ar] Armstrong W.W., Dependency Structures of Database Relationships. Information Processing 74, Holland Publ. Co., 1974, 580-583.

[BB] Beeri C., Bernstein P. A., Computational problems related to the design of normal form relation schemes. ACM Trans. on Database Syst. 4, 1, 1979, 30-59.

[BDFS] Beeri C., Dowd M., Fagin R., Staman R., On the Stucture of Armstrong relations for Functional Dependencies. J. ACM 31, 1, 1984, 30-46.

[DFK] Demetrovics J., Furedi Z., Katona G.O.H., Minimum matrix representation of closure operations. Discrete Applied Mathematics, North Holland 11, 1985, 115-128.

[DK1] Demetrovics J., Katona G.O.H., Extremal combinatorial problems of databases. In: MFDBS'87, 1st Symposium on Mathematical Fundamental of Database Systems, Dresden, GDR, January, 1987, Lecture Notes in Computer Science, Berlin: Spinger 1987, 99-127.

[DK2] Demetrovics J., Katona G.O.H., A survey of some combinatoria! results concerning functional dependencies in database relations. Annals of Mathematics and Artificial Intelligence, 7, 1993, 63-82.

[DLM] Demetrovics J., Libkin L., Muchnik I.B. Functional dependencies in relational databases: A lattice point of view. Discrete Applied Mathematics, North Holland, 40, 1992, 155-185.

[DHLM] Demetrovics J., Hencsey G., Libkin L., Muchnik I.B., Normal Form Relation Schemes: A New Charaterization. Acta Cybernetica, Hungary, 10, 3, 1992, 141-153.

[DT1] Demetrovics J., Thi V.D., On algorithms for generating Armstrong relations and inferring functional dependencies in the Relational datamodel. Computers and Mathematics with Applications , Greate Britain, 26, 4, 1993, 43-55.

[DT2] Demetrovics J., Thi V.D., Some results about normal forms for functional dependencies in the relational data model. Discrete Applied Mathematics, North Holland, 69, 1996, 61-74.

[DT3] Demetrovics J., Thi V.D., Describing Candidate Keys by Hypergraphs. Computer and Artificial Intelligence , V.18, N. 2, 1999, 191-207.

[DT4] Demetrovics J., Thi V.D., Family of functional dependencies and its equivalent descriptions. J. Computer and Math with Application, Great Britain, 29, 4, 1995, 101-109.

[Li] Libkin L., Direct Product Decomposition of Lattices, Closures and Relation Schemes. Discrete Mathematics, North Holland, 112, 1993, 119-138

[MR] Mannila H., Raiha K. J., On The Complexity of Inffering Functional Dependencies. Discrete Applied Mathematics, North Holland, 40, 1992, 237-243.

[RG] Ramakrishnan R., Gehrke J., Database Management Systems. The McGraw - Hill, 2000.

[Ul] Ullman J., Principles of Database and Knowledge Base Systems, VoL 1. Computer Science Press, 1988.

Received September, 2003

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János